DENSO

BHT-BASIC

Programmer's Manual

Preface

This manual describes the syntax and development procedure of BHT-BASIC 3.5 which is a programming language for developing application programs of the BHT-3000/BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500 Bar Code Handy Terminals.

It is intended for programmers who already have some experience in BASIC programming. For the basic description about the BASIC language, refer to documentations concerning Microsoft BASIC® or QuickBASIC®. For the details about Windows™, refer to the Microsoft Windows documentations.

How this book is organized

This manual is made up of 18 chapters and appendices.

Chapter 1. Software Overview for the BHT

Surveys the software structure of the BHT and introduces the programs integrated in the ROM and the language features of BHT-BASIC.

Chapter 2. Development Environment and Procedures

Describes hardware and software required for developing application programs and the developing procedure.

Chapter 3. Program Structure

Summarizes the basic structure of programs and programming techniques, e.g., program chaining and included files.

Chapter 4. Basic Program Elements

Describes the format of a program line, usable characters, and labels.

Chapter 5. Data Types

Covers data which the programs can handle, by classifying them into data types--constants and variables.

Chapter 6. Expressions and Operators

Surveys the expressions and operators to be used for calculation and for handling concatenated character strings. The operators connect, manipulate, and compare the expressions.

Chapter 7. I/O Facilities

Defines I/O facilities and describes output to the LCD, input from the keyboard, and control for the timer, beeper, and other I/Os by the statements and functions.

Chapter 8. Files

Describes data files and device files.

Chapter 9. Event Polling and Error/Event Trapping

Describes the event polling and two types of traps: error traps and event (of keystroke) traps supported by BHT-BASIC.

Chapter 10. Sleep Function

Describes the sleep function.

Chapter 11. Resume Function

Describes the resume function.

Chapter 12. Power-related Features

Describes low battery warning, the prohibited simultaneous operation of the beeper/illumination LED (or laser source)/LCD backlight, the wakeup, and remote wakeup.

Chapter 13. LCD Backlight Function

Describes the LCD backlight function

Chapter 14. Statement Reference

Describes the statements available in BHT-BASIC, including the error codes and messages.

Chapter 15. Function Reference

Describes the functions available in BHT-BASIC, including the error codes and messages.

Chapter 16. Extended Functions

Describes the extended functions newly added in the BHT-7000/BHT-7500.

Chapter 17. Spread Spectrum Communications (available with the BHT-7500S)

Summarizes the spread spectrum communication system that may be configured with the BHT-7500S. This chapter also explains wireless-related statements and the function library SS.FN3 to be used in wireless communications programming.

Chapter 18. TCP/IP

Surveys the socket application program interface (API) and FTP client. This chapter also describes the two function libraries--SOCKET.FN3 and FTP.FN3, which are built in the BHT-7500S for providing BHT-BASIC programs with access to a subset of the TCP/IP family of protocols over wireless communications devices.

Appendix A: Error Codes and Error Messages

B: Reserved Words

C: Character Sets

D: I/O Ports

E: Key Number Assignment on the Keyboard

F: Memory Area

G: Handling Space Characters in Downloading

H: Programming Notes

I: Program Samples

J: Quick Reference for Statements and Functions

K: Unsupported Statements and Functions

■ Notational Conventions Used in This Book

Several notational conventions are used in this book for the sake of clarity.

1. Reserved words are printed in UPPERCASE. These are BHT-BASIC's keywords. You should not use them as label names or variable names.

   Example: `CHAIN`, `GOSUB`, and `ABS`

2. Parameters or arguments which should be specified in the statements or functions are expressed in *italics*.

   Example: *`characode`* and *`onduration`*

3. Items enclosed in square brackets [ ] are optional, which can be omitted.

   Example: `[`*`commonvariable`*`]`

4. Items enclosed in braces { } and separated by vertical bars | represent alternative items. You should choose either item.

   Example: `CURSOR {ON|OFF}`

5. An ellipsis . . . indicates that you can code the previous item described in one line two or more times in succession.

   Example: `READ` *`variable`*`[,`*`variable`*`...]`

6. Hexadecimal values are followed by h. In many cases, hexadecimal values are enclosed with parentheses and preceded by decimal values.

   Example: `65 (41h)` and `255 (FFh)`

   In program description, hexadecimal values are preceded by &H.

   Example: `&H41` and `&HFF`

7. Programs make no distinction between uppercase and lowercase letters, except for character string data.

   The uppercase-lowercase distinction used in this manual is intended to increase the legibility of the statements. For example, reserved words are expressed in uppercase; label names and variable names in lowercase. In practical programming, it is not necessary to observe the distinction rules used in this manual.

   The examples below are regarded as the same.

   Example 1: `&HFFFF`, `&hffff`, and `&hFFFF`
   Example 2: `A AND B`, `a and b`, and *`a`* `AND b`
   Example 3: `PRINT STR$(12)`, `Print Str$(12)`, and `print str$(12)`

■ Icons Used in This Book

Statements and functions unique to BHT-BASIC.

■ Syntax for the Statement Reference and Function Reference

The syntax in programming is expressed as shown in the example below.

For the `INPUT` statement

Syntax: `INPUT [;]["prompt"{,|;}]variable`

According to the above syntax, all of the following samples are correct:

```
INPUT;keydata
INPUT keydata
INPUT "input =",keydata
INPUT;"input =";keydata
```

■ Technical Terms Used in This Manual

Compiler and Interpreter

The BHT-BASIC Compiler, which is a development tool, is expressed as Compiler.

The BHT-BASIC Interpreter, which runs in the BHT, is expressed as Interpreter.

Source Program and Object Program (User Program)

Generally, a source program is translated into an object program by a compiler. This manual calls an object program a user program.

Easy Pack

Easy Pack is an application program suitable for data collection. Listed below are the versions and memories in which Easy Pack is to be stored. For details about each version of Easy Pack, refer to the respective manual shown below.

BHT Series	Version of Easy Pack	Memory	Refer to:
BHT-3000	Easy Pack 4.1	ROM	"BHT-3000 User's Manual"
BHT-4000	Easy Pack 4.2	User area of RAM	"BHT-4000 User's Manual"
BHT-5000 BHT-6000 BHT-6500	Easy Pack Pro	User area of RAM or flash ROM	"Easy Pack Pro User's Manual"

BHT and CU

This manual expresses the BHT-3000/BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500 as BHT. To designate each of them, it expresses the series or model as listed below.

Series	Model	RAM Capacity	Flash ROM Capacity
BHT-3000	BHT-3041	128KB	—
	BHT-3045	512KB	—
BHT-4000	BHT-4082	256KB	—
	BHT-4086	768KB	—
	BHT-4089	2048KB	—
BHT-5000	BHT-5071	128KB	512KB
	BHT-5075	512KB	512KB
	BHT-5077	1024KB	512KB
	BHT-5079	2048KB	512KB
BHT-6000	BHT-6045	512KB	512KB
	BHT-6047	512KB	1024KB
	BHT-6049	512KB	2048KB
BHT-6500	BHT-6505	512KB	512KB
	BHT-6509	2048KB	512KB
BHT-7000	BHT-7064	512KB	2048KB
BHT-7500	BHT-7508	1024KB	8192KB
	BHT-7508S	512KB	8192KB

In the same way as above, the CU-3000/CU-4000/CU-5000/CU-6000/CU-7000 are expressed as CU.

■ Abbreviations

ANK	AlphaNumerics and Katakana
BASIC	Beginners All purpose Symbolic Instruction Code
BCC	Block Check Character
BHT	Bar code Handy Terminal
CTS (CS)	Clear To Send (RS-232C signal control line)
CU	Communications Unit
I/F	Interface
I/O	Input/Output
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
MOD	Modulo
MS-DOS	Microsoft-Disk Operating System
RAM	Random Access Memory
ROM	Read Only Memory
RTS (RS)	Request To Send (RS-232C signal control line)
VRAM	Video RAM

■ Related Publications

For BHT-3000	BHT-3000 User's Manual
For BHT-4000	BHT-4000 User's Manual
For BHT-5000	BHT-5000 User's Manual Multilink Transfer Utility Guide
For BHT-6000	BHT-6000 User's Manual
For BHT-6500	BHT-6500 User's Manual
For BHT-7000	BHT-7000 User's Manual
For BHT-7500	BHT-7500 User's Manual
For all of the BHTs	Transfer Utility Guide
For BHT-6000/BHT-6500/BHT-7000/BHT-7500	Ir-Transfer Utility C Guide Ir-Transfer Utility E Guide
For BHT-4000R/BHT-5000R	BHT-BASIC Programmer's Manual For Radio Communications

■ Screen Indication

The lettering in the screens of the BHT and host computer in this manual is a little different from that in the actual screens. File names used are only for description purpose, so they will not appear if you have not downloaded files having those names to the BHT.

Chapter 1
Software Overview for the BHT

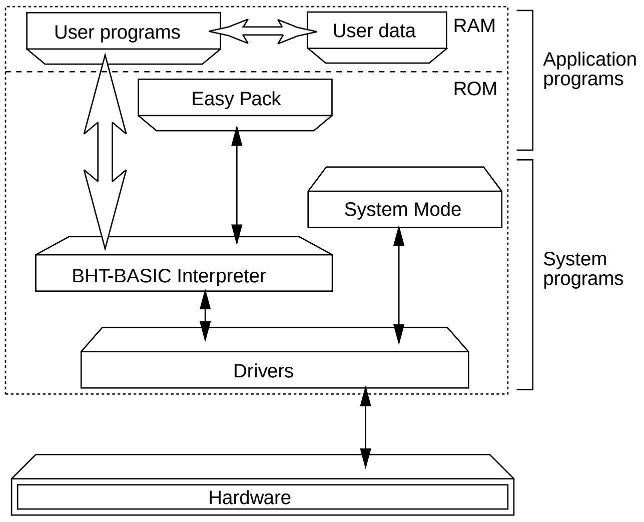
CONTENTS

1.1 Software Overview

1.1.1 Software Structure of the BHT

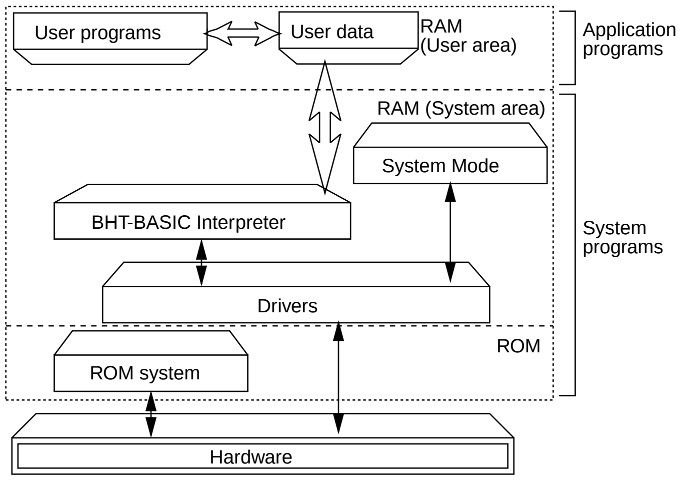
The structure of software for the BHT is shown below.

■ BHT-3000



When downloaded, user programs will be stored in the RAM. Other programs reside in the ROM.

■ BHT-4000

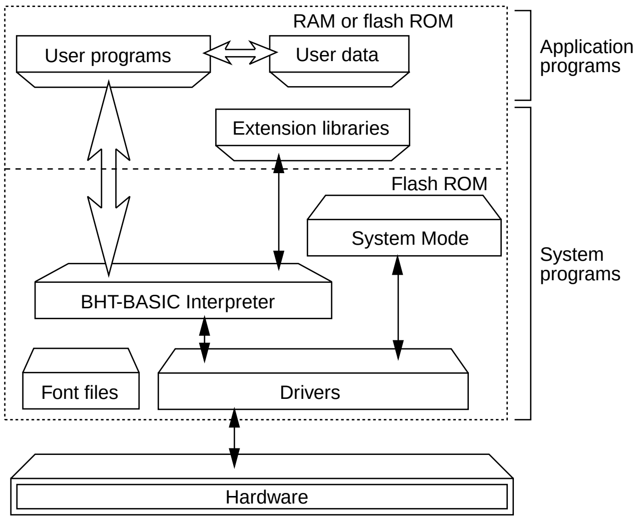


Of all the system programs, the drivers, BHT-BASIC Interpreter, and System Mode will be stored in the system area of the RAM when downloaded. The ROM system resides in the ROM.

User programs downloaded will be stored in the user area of the RAM.

(For the downloading procedure of the system programs, refer to the "BHT-4000 User's Manual.")

■ BHT-5000/BHT-6000/BHT-6500



The BHT-5000/BHT-6000/BHT-6500 has a flash ROM and RAM. In the flash ROM reside the drivers, BHT-BASIC Interpreter, System Mode, and font files. Extension libraries and user programs will be stored in the user area of the RAM (or in the flash ROM) when downloaded.

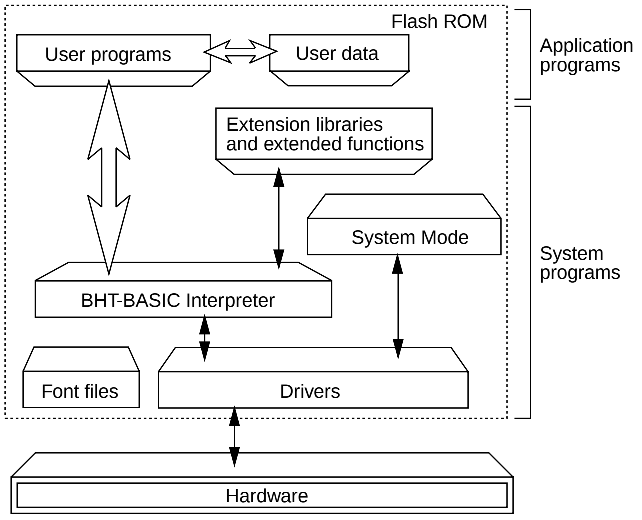
NOTE

Unlike the RAM, the flash ROM requires no power for retaining stored files. Therefore, leaving the BHT with no rechargeable battery cartridge or dry batteries loaded will not damage those files stored in the flash ROM while it may damage those files in the RAM.

The flash ROM has the following restrictions so that you cannot use it like the RAM:

- The frequency of rewriting operations is limited to approx. 100,000 times.
- In application programs, you cannot write data onto the flash ROM.

■ BHT-7000/BHT-7500



The BHT-7000/BHT-7500 has a flash ROM and RAM. All of the system programs, user programs, extension libraries, and extended functions are stored in the flash ROM. The RAM is used to run those programs efficiently.

□ System Programs

Drivers

A set of programs which is called by the BHT-BASIC Interpreter or System Mode and directly controls the hardware. The drivers include the Decoder Software used for bar code reading.

BHT-BASIC Interpreter

Interprets and executes user programs and Easy Pack commands.

System Mode

Sets up the execution environment for user programs or Easy Pack.

ROM System (BHT-4000)

Required for downloading the system programs listed above to the BHT-4000.

Extension Library (BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500)

A set of programs which extends the function of the BHT-BASIC to enable the following:

- Displaying ruled lines on the LCD
- Transmitting/receiving files by using the X-MODEM and Y-MODEM protocols

These extension programs are stored in files having an FN3 extension, in each file per function. You should download a xxxx.FN3 file containing the necessary function from the BHT-BASIC Extension Library (sold separately) to the user area.

Extended Functions (BHT-7000/BHT-7500)

A set of functions integrated in system programs, which extends the function of the BHT-BASIC. No downloading is required for those functions since they are integrated in System. For details, refer to Chapter 16, "Extended Functions."

NOTE The extension libraries for the BHT-5000/BHT-6000/BHT-6500 are different from those for the BHT-7000/BHT-7500. Use extension libraries suited for your BHT.

□ Application Programs

User Programs

User-written object programs which are ready to be executed.

Easy Pack

Application program used for bar code data collection.

1.1.2 Overview of BHT-BASIC

With BHT-BASIC, you can customize application programs for meeting your specific needs as given below.

- Retrieving products names, price information, etc. in a master file.
- Making a checking procedure more reliable with check digits in bar code reading.
- Improving the checking procedure by checking the number of digits entered from the keyboard.
- Calculating (e.g., subtotals and totals).
- Supporting file transmission protocols (or transmission procedures) suitable for host computers and connected modems.
- Downloading master files.
- Supporting a program capable of transferring control to several job programs depending upon conditions.

1.2 BHT-BASIC

1.2.1 Features

BHT-BASIC is designed as an optimal programming language in making application programs for the bar code handy terminal BHT, and to enable efficient program development, with the following features:

■ Syntax Similar to Microsoft™ BASIC

BHT-BASIC uses the BASIC language which is the most widely used one among the high-level languages. The syntax of BHT-BASIC is as close as possible to that used in Microsoft BASIC (MS-BASIC).

■ No Line Numbers Required

BHT-BASIC requires no line number notation. You can write a branch statement with a label instead of a line number so that it is possible to use cut and paste functions with an editor in developing source programs, thus facilitating the use of program modules for development of other programs.

■ Program Development in Windows95/98 or WindowsNT/Windows2000

You may develop programs with BHT-BASIC on those computers operating on Windows95/98 or WindowsNT4.0/Windows2000.

■ Advantages of the Dedicated Compiler

The dedicated compiler outputs debugging information including cross reference lists of variables and labels, enabling the efficient debugging in program development.

The Compiler assigns variables to fixed addresses so that it is not necessary for the Interpreter to allocate or release memories when executing user programs, making the execution time shorter.

■ Program Compression by the Dedicated Compiler

The Compiler compresses a source program into the intermediate language to produce an object program (a user program).

(When a compiled user program is downloaded to the BHT, the BHT packs a pair of ASCII bytes into a single byte by converting each byte into a 4-bit hexadecimal number for more efficient use of the memory area in the BHT.)

1.2.2 What's New in BHT-BASIC 3.5 Upgraded from BHT-BASIC 3.0?

Based on BHT-BASIC 3.0, BHT-BASIC 3.5 newly supports the following functions:

[1] Compiler

■ Object linkage editor, Linker

While BHT-BASIC 3.0 Compiler compiles a single source program into a single user program, BHT-BASIC 3.5 Compiler can convert more than one source program into individual object programs (intermediate code files for a user program) and then combine them together through Linker to build a user program. With Linker, you may use existing object programs for development of user programs.

■ Libraries

The Librarian allows you to build libraries out of object files resulting from compiling, which makes it easier to use existing application programs. This facilitates the use of existing application programs for development of other programs.

■ Projects

BHT-BASIC 3.5 has added a concept of Project that makes it easier to use multiple source programs for producing a user program.

[2] Statements

■ Added statements

Based on BHT-BASIC 3.0, BHT-BASIC 3.5 newly supports several statements for making distinction between global variables and local variables, and for defining functions and constants.

Newly added statements

CALL	Calls a SUB function in addition to an FN3 function.
CONST	Defines symbolic constants to be replaced with labels.
DECLARE	Declares user-created function `FUNCTION` or `SUB` externally defined.
FUNCTION…END FUNCTION	Names and defines user-created function `FUNCTION`.
GLOBAL	Declares one or more work variables or register variables defined in a file, as global variables.
PRIVATE	Declares one or more work variables or register variables defined in a file, as local variables.
SUB...END SUB	Names and defines user-created function SUB.

■ Defining and declaring user-defined functions more easily

BHT-BASIC 3.5 has added FUNCTION…END FUNCTION, SUB...END SUB, and DECLARE statements. With the former two, you may easily define your own functions—FUNCTION and SUB. With the latter one, you may declare FUNCTION and SUB functions which are defined in any other source files.

■ Distinction between local variables and global variables (defined by PRIVATE and GLOBAL statements)

BHT-BASIC 3.5 makes distinction between local variables and global variables to restrict the access to the variables.

Both local variables and global variables may be defined for work variables and register variables. Local variables can only be accessed by any routine in the file where that variable is defined. Global variables can be accessed by any routine in a program.

However, variables used without declaration inside FUNCTION or SUB function are regarded as local variables and can be accessed only within that function.

Since local variables are restricted in access, you can define them with the same name in different files.

■ Defining constants

BHT-BASIC 3.5 can define constants.

1.3 Program Development and Execution

BHT-BASIC consists of Compiler and Interpreter.

1.3.1 Compiler

BHT-BASIC 3.5 Compiler consists of the following Compiler, Linker and Librarian:

■ Compiler

Compiler, which is one of the development tools, compiles source programs written on a PC into the resulting "object files."

It checks syntax of source programs during compilation and makes an error file if any syntax error is found.

■ Linker

Linker, which is one of the development tools, combines object files (translated by Compiler) together to build a "user program" in the intermediate language.

If linking does not end normally, Linker makes an error file.

■ Librarian

Librarian, which is one of the development tools, builds "library files" out of object files translated by Compiler.

If Librarian does not end normally, it makes an error file.

1.3.2 Interpreter

Interpreter interprets and executes a user program downloaded to the BHT, statement by statement.

Chapter 2
Development Environment and Procedures

CONTENTS

2.1 Overview of Development Environment

The following hardware and software are required for developing user programs:

2.1.1 Required Hardware

■ Personal computer

Use a computer operating with Windows95/98 or WindowsNT4.0/Windows2000.

■ BHT (Bar code handy terminal)

Any of the following BHTs is required:

- BHT-3000
- BHT-4000
- BHT-5000
- BHT-6000
- BHT-6500
- BHT-7000
- BHT-7500

■ CU (Optical communications unit)

For optical communications, any of the following CUs is required. Note that no CU is required if the BHT is directly connected with the host computer via the direct-connect interface.

• CU-3000	(for BHT-3000)
• CU-4000	(for BHT-4000)
• CU-5000	(for BHT-5000)
• CU-6000	(Option for BHT-6000/BHT-6500. Required if the host computer has no IR interface port.)
• CU-7000	(Option for BHT-7000/BHT-7500. Required if the host computer has no IR interface port.)

■ RS-232C interface cable

This cable connects the CU with the personal computer.

NOTE The RS-232C interface cable should have the connector and pin assignment required by the personal computer.

(For information about the connector configuration and pin assignments of the CU, refer to the BHT's User's Manual.)

2.1.2 Required Software

• **OS**	**Windows95/98 or WindowsNT4.0/Windows2000**	
• **Editor**		
• **BHT-BASIC 3.5 Compiler**	**BHTC35W.EXE**	**(Integrated environment manager)**
	BHT35CPL.DLL	**(Compiler)**
	BHT35LNK.DLL	**(Linker)**
	BHT35LIB.DLL	**(Librarian)**
	BHTC35W.MSG	**(Error message file)**
• **Transfer Utility (option)**	**TU3.EXE**	**(MS-DOS–based)**
	TU3W.EXE	**(16-bit Windows-based)**
	TU3W32.EXE	**(Windows-based)**
• **Ir-Transfer Utility C (option)**	**IT3C.EXE**	**(MS-DOS–based)**
	IT3CW32.EXE	**(Windows-based)**
• **Ir-Transfer Utility E (option)**	**IT3EW32.EXE**	**(Windows-based)**

Transfer Utility, Ir-Transfer Utility C, or Ir-Transfer Utility E is an essential tool for downloading user programs to the BHT.

Each of the BHT-BASIC Compiler, Transfer Utility, Ir-Transfer Utility C, Ir-Transfer Utility E is optionally provided in a CD or floppy disk.

NOTE Prepare editor versions which are operable with the personal computer on which user programs are to be developed.

For the manufacturers and models of computers to which Transfer Utility, Ir-Transfer Utility C, or Ir-Transfer Utility E is applicable, refer to the “Transfer Utility Guide,” “Ir-Transfer Utility C Guide,” or “Ir-Transfer Utility E Guide,” respectively.

2.2 Overview of Developing Procedures

2.2.1 Developing Procedures

The program developing procedures using BHT-BASIC 3.5 are outlined below.

- Making source programs

  Make source programs with an editor according to the syntax of BHT-BASIC.

- Producing a user program (compiling and linking)

  Compile the source programs into object programs by BHT-BASIC Compiler. Then combine those object programs or libraries (made up by Librarian) together through Linker to produce a user program in the intermediate language format.

- Downloading the user program

  Download the user program to the BHT by using Transfer Utility/Ir-Transfer Utility C/ Ir-Transfer Utility E.

- Executing the user program

  Execute the user program on the BHT.

2.2.2 Functions of BHT-BASIC 3.5

BHT-BASIC 3.5 contains Compiler, Linker, and Librarian whose functions are listed below.

Functions of Compiler	Description
Syntax check	Detects syntax errors in source programs.
Output of object files	Translates source programs into object files and outputs them.
Output of debug information	Outputs list files and debug information files required for debugging.

Functions of Linker	Description
Output of a link map file	Outputs a symbol table along with its memory address.
Output of a user program	Integrates more than one object program or library to produce a user program in the intermediate language format. When downloaded to the BHT by Transfer Utility/Ir-Transfer Utility C/Ir-Transfer Utility E, the user program will be compressed into programs that the Interpreter can translate.

Functions of Librarian	Description
Output of a library	Builds a library out of multiple object files. The library is a collection of object files that Linker will use.

2.3 Writing a Source Program

2.3.1 Writing a Source Program by an Editor

To write a source program, use an editor designed for operating environments where the BHT-BASIC 3.5 Compiler will execute. The default editor is Windows Notepad.

TIP To write a source program efficiently, use of a commercially available editor is recommended. For the operation of such an editor, refer to the instruction manual for the editor.

2.3.2 Rules for Writing a Source Program

When writing a source program according to the syntax of BHT-BASIC 3.5, observe the following rules:

- A label name should begin in the 1st column.

```
ABC
```

```
2000
```

- A statement should begin in the 2nd or the following columns.

```
     PRINT
     FOR I=1 TO 100 : NEXT I
```

- One program line should be basically limited to 512 characters (excluding a CR code) and should be ended with a CR code (by pressing the carriage return key).

  If you use an underline (_) preceding a CR code, however, one program line can be extended up to 8192 characters. For statements other than the `PRINT`, `PRINT#`, and `PRINT USING` statements, you may use also a comma (,) preceding a CR code, instead of an underline.

- Comment lines starting with a single quotation mark (') and those with a REM should have the following description rules each.

  A single quotation mark (') can be put starting from the 1st or the following columns, or immediately following any other statement.

  A REM should be put starting from the 2nd column or the following columns. To put a REM following any other statement, a colon (:) should precede the REM.

```
‘Comment
     CLS    ‘Comment
```

```
     REM    Comment
     CLS    :REM   Comment
```

- It is necessary to end the IF statement with an END IF or ENDIF, since the IF statement will be treated as a block-structured statement.

```
     IF a$ = “Y” OR a$ = “y” THEN
            GOTO SUB12
     END IF
```

- The default number of characters for a non-array string variable is 40; that for an array string variable is 20.

  Specifying the DIM or DEFREG statement allows a single string variable to treat 1 through 255 characters.

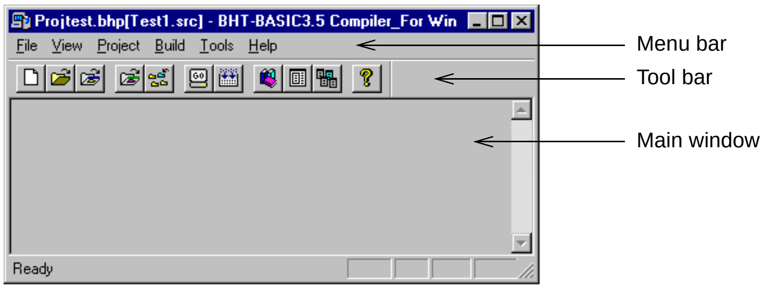
```
     DIM b$[255]
     DIM c$(2,3)[255]
     DEFREG d$[255]
     DEFREG e$(2,3)[255]
```

NOTE BHT-BASIC does not support some of the statements and functions used in Microsoft BASIC or QuickBASIC. For details, refer to Appendix K, “Unsupported Statements and Functions.”

2.4 Producing a User Program

2.4.1 Starting the BHT-BASIC 3.5 Compiler

Start the Compiler, e.g., by choosing the "BHTC35W.EXE" from the Windows Explorer or the "BHT-BASIC 3.5" registered to the Start menu.


Projtest.bhp[Test1.src] - BHT-BASIC3.5 Compiler_For Win
File View Project Build Tools Help
Ready
Menu bar
Tool bar
Main window

The BHT-BASIC 3.5 Compiler supports the following menus and icons which provide quick ways to do tasks:

Menus	Commands	Icons	Functions
File	New	🗋	Creates a new project.
	Open	📂	Opens an existing file.
	Close		Closes the active file.
	Open Project	📂 (Yellow)	Opens an existing project.
	Close Project		Closes the active project.
	Exit		Quits the BHT-BASIC 3.5 Compiler.
View	Toolbar		Shows or hides the toolbar.
	Status Bar		Shows or hides the status bar.
	Clear Screen		Clears the screen.
Project	Select File	📂 (Red)	Selects or deletes a file in the active project.
	Add File	▦	Adds one or more files to the active project.
Build	Compile	▣	Compiles one or more active files (or active project) to produce an object file(s).
	Build	▦	Compiles one or more active files (or active project) and then links them to produce a user program.
Tools	Options	▣	Sets compiling options and linking options.
	Run Editor	▤	Runs the editor.
	Set Editor	▦	Selects the editor you want to run.
Help	About BHT-BASIC 3.5	?	Displays the program information, version number and the copyright.

2.4.2 Outline of User Program or Library Production Procedure

Unlike the BHT-BASIC 3.0 Compiler that converts a single source program into a user program (file named XXX.PD3), the BHT-BASIC 3.5 Compiler converts source programs into object programs (files named XXX.OBJ) and then links those object programs to produce a user program (XXX.PD3). A sequence of the compiling and linking processes is called "Build."

The BHT-BASIC 3.5 Compiler can also build a library (XXX.LIB). You may select whether you build a user program or library on the Project Configuration Files dialog box.

You may build a user program or library out of either multiple files or a single file (as in the BHT-BASIC 3.0 Compiler).

Note that to build a library out of a single source file, you need to create a project file for a single source file.

[1] Building a user program out of a single source program file

What follows is a general procedure for building a user program out of a single source program file.

(1) Designate a file that you want to use. (For details, refer to Subsection 2.4.3.1, "Designating a single source file.")

(2) Build a user program out of the designated file. (For details, refer to Subsection 2.4.4, [3], "Building.")

[2] Building a library out of a single source file, or building a user program or library out of multiple source files

What follows is a general procedure for building a library out of a single source file or for building a user program or library out of multiple source files.

(1) Designate a project that you want to use. (For details, refer to Subsection 2.4.3.2, "Designating a project file.")

(2) Build a user program or library out of the designated project. (For details, refer to Subsection 2.4.4, [3], "Building.")

2.4.3 Designating a Single Source File or a Project File

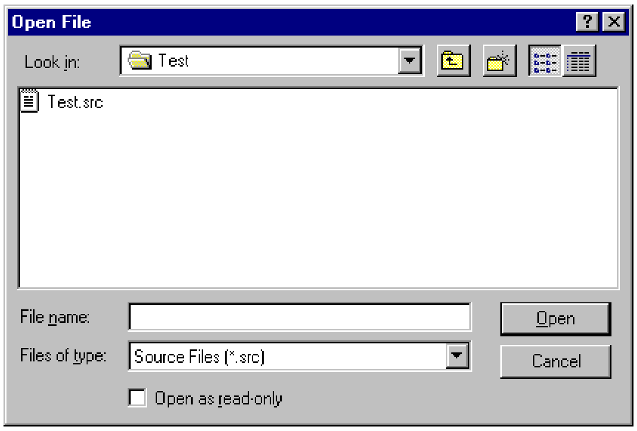
2.4.3.1 Designating a single source file

Just as in the conventional BHT-BASIC 3.0 Compiler, you may designate a single source file to build a user program or library.

[1] Select a source file

(1) In any of the following methods, display the Open File dialog box shown below:

- From the File menu, choose the Open command.
- Click the open file button in the toolbar.
- While holding down the Ctrl key, press the O key.



(2) Select a source file you want to use and then click the Open button.

Then the source file opens.

(3) Proceed to Section 2.4.4, "Compiling and Building."

2.4.3.2 Designating a project file

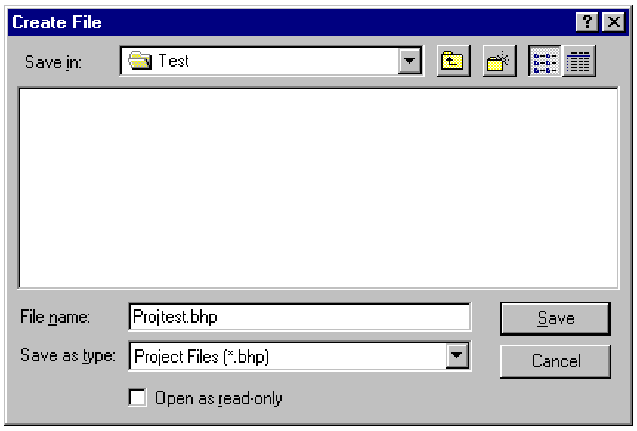
To build a library out of a single source file or to build a user program or library out of multiple source files, you need to create a project file (described in [1] later) or select an existing project file (in [2]).

You may add files or delete existing files to/from the designated project file (described in [3] and [4], respectively).

[1] Create a new project

(1) In any of the following methods, display the Create File dialog box shown below:

- ■ From the File menu, choose the New command.
- ■ Click the new file button 🗋 in the toolbar.
- ■ While holding down the Ctrl key, press the N key.



File name: Projtest.bhp
Save as type: Project Files (*.bhp)

(2) Designate a project file you want to create (Projtest.bhp in this example), and then click the Save button.

If you create a project file having the same name as one already used, the warning message dialog box will appear. If you want to overwrite, click the OK button; if you do not, click the Cancel button to quit the project creating procedure.

(3) The Add File(s) dialog box appears. Into the newly created project, you need to put files which should configure the project, according to the statements given in [3], "Add files to a project file."

[2] Select an existing project file

You may select an existing project file in the Select Project File dialog box or in the Open File dialog box.

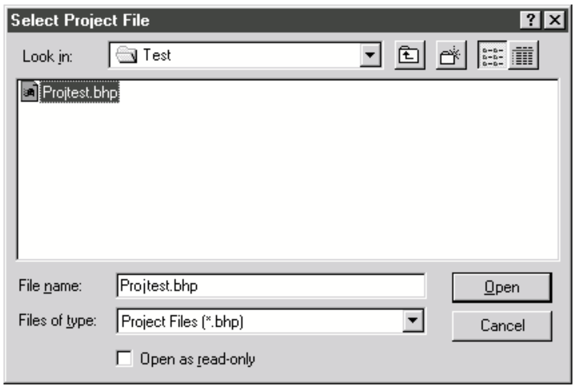
Selecting in the Select Project File dialog box

(1) In any of the following methods, display the Select Project File dialog box shown below:

■ From the File menu, choose the Open Project command.

■ Click the open project button (yellow) in the toolbar.

■ While holding down the Ctrl key, press the P key.



Projtest.bhp

File name: Projtest.bhp

Files of type: Project Files (*.bhp)

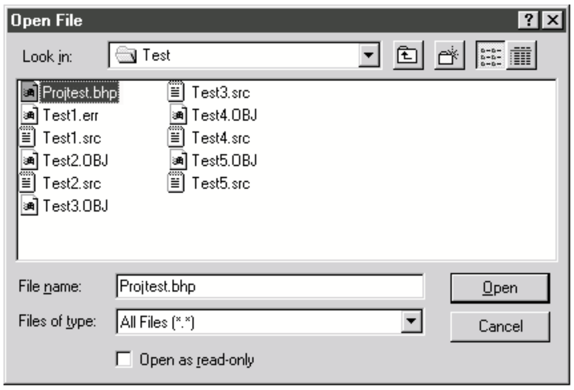
(2) Select an existing project file you want to use (Projtest.bhp in this example), and then click the Open button.

(3) Proceed to Section 2.4.4, "Compiling and Building."

Selecting in the Open File dialog box

(1) Display the Open File dialog box, referring to Subsection 2.4.3.1, [1].

(2) Select an existing project file you want to use (Projtest.bhp in this example), and then click the Open button.



Projtest.bhp
Test1.err
Test1.src
Test2.OBJ
Test2.src
Test3.OBJ
Test3.src
Test4.OBJ
Test4.src
Test5.OBJ
Test5.src

File name: Projtest.bhp

Files of type: All Files (*.*)

(3) Proceed to Section 2.4.4, "Compiling and Building."

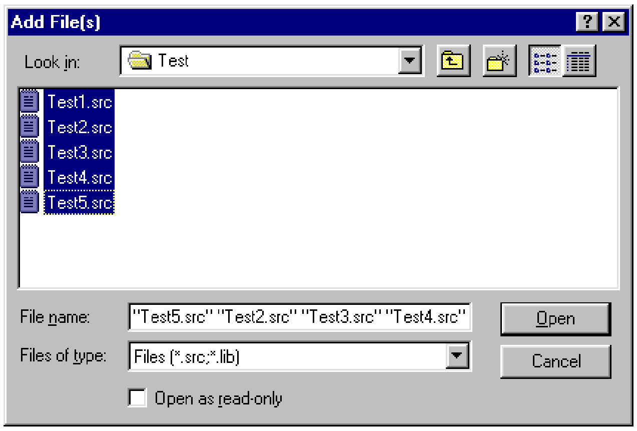
[3] Add files to a project file

You may add one or more source files and libraries to a project file.

(1) Create a new project (Refer to [1] in this subsection) or select an existing project file to which you want to add files (Refer to [2] in this subsection).

(2) In either of the following methods, display the Add File(s) dialog box shown below:

- ■ From the Project menu, choose the Add File command.
- ■ Click the add file button in the toolbar.



File name: "Test5.src" "Test2.src" "Test3.src" "Test4.src"
Files of type: Files (*.src;*.lib)

(3) Select files you want to add to the active project file and then click the Open button.

(4) The Project Configuration Files dialog box will appear which lists files in the project. For details about the Project Configuration Files dialog box, refer to [4], "Select files in the active project" given later.

[4] Select files in the active project

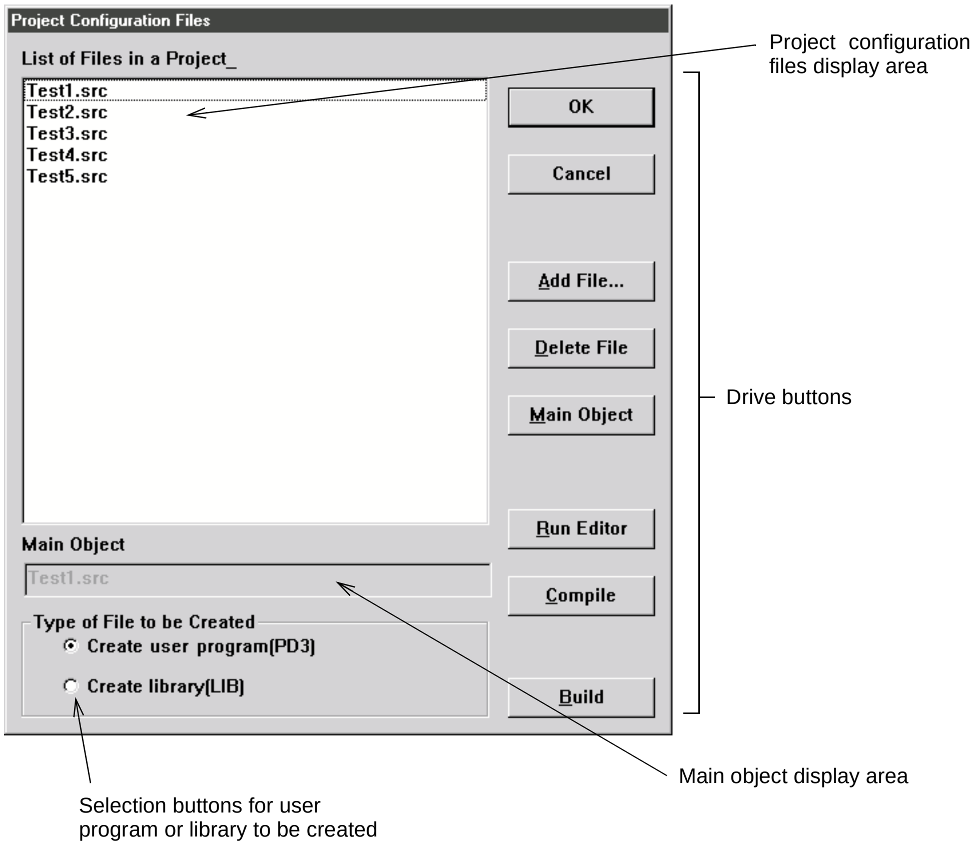
From files existing in the active project, you may select files that you want to compile or build.

(1) In either of the following methods, display the Project Configuration Files dialog box shown below:

- From the Project menu, choose the Select File command.
- Click the select file button (red) in the toolbar.

TIP The Project Configuration Files dialog box will appear also following the new project creation process (see [1] earlier) or the file addition process to an existing project (see [3] earlier).

(2) Select files you want to compile or build.



Project Configuration Files

List of Files in a Project_

Test1.src
Test2.src
Test3.src
Test4.src
Test5.src

Main Object: Test1.src

Type of File to be Created: [✓ Create user program(PD3)] Create library(LIB)

(3) In the Project Configuration Files dialog box are the following display areas and buttons from which you may also select a user program or library to be built, may start compiling or building, and may run the editor, as well as adding or deleting files to/from the active project.

- *List of Files in a Project*

  This display area shows a list of files which configures the active project. The filenames are displayed as a relative path.

- *Main Object*

This display area shows the name of a main object in a user program if you have selected "User program (PD3)" with the "Type of File to be Created" selection button. If you have selected "Create library (LIB)," nothing will appear on this area.

- *Type of File to be Created*

Lets you select whether you create a user program (PD3) or library (LIB).

- *Add File button*

Adds the currently selected files to the active project. (Refer to “[3] Add files to a project file.”)

- *Delete File button*

Deletes the currently selected file(s) from the active project.

- *Main Object button*

Specifies the currently selected file as a main object if you have selected "User program (PD3)" with the “Type of File to be Created” selection button. A library cannot be specified as a main object.

This button will be disabled if more than one file is selected or “Create library (LIB)” is selected with the “Type of File to be Created” selection button.

- *Run Editor button*

Opens a file currently selected by the editor.

- *Compile button*

Compiles currently selected source files into object files.

- *Build button*

Builds a user program out of the active project.

2.4.4 Compiling and Building

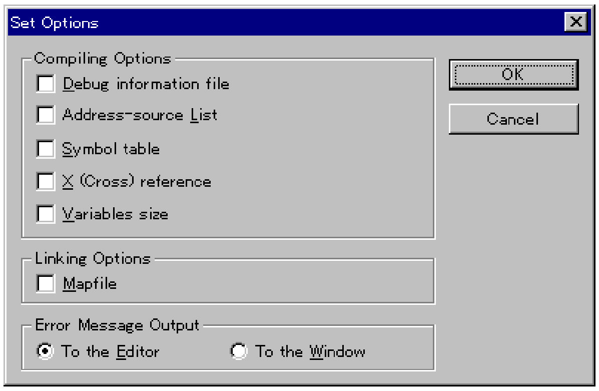
First specify the options and then proceed to the compiling or building process.

[1] Specifying the compiling and linking options

(1) In either of the following methods, display the Set Options dialog box shown below:

■ From the Tools menu, choose the Options command.

■ Click the option button in the toolbar.



Set Options

Compiling Options
- ☐ Debug information file
- ☐ Address-source List
- ☐ Symbol table
- ☐ X (Cross) reference
- ☐ Variables size

Linking Options
- ☐ Mapfile

Error Message Output
- ● To the Editor
- ○ To the Window

(2) Select the check boxes of the options you want to specify.

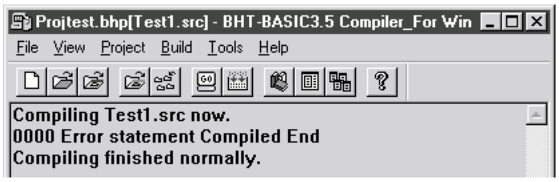
For details about the options, refer to Subsection 2.4.7.

[2] Compiling

In any of the following methods, compile the currently selected source file(s) into an object file(s):

- From the Build menu, choose the Compile command.
- In the Project Configuration Files dialog box, click the Compile button. (For details about the Project Configuration Files dialog box, refer to Subsection 2.4.3.2, [4].)
- Click the compile start button in the toolbar.
- While holding down the Ctrl key, press the G key.

If compiling ends normally, the screen shown below will appear.


Projtest.bhp[Test1.src] - BHT-BASIC3.5 Compiler_For Win
File View Project Build Tools Help

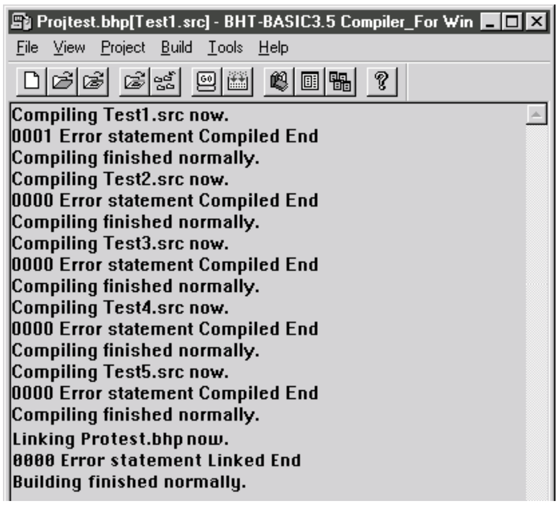
```
Compiling Test1.src now.
0000 Error statement Compiled End
Compiling finished normally.
```

[3] Building

In any of the following methods, build a user program or library out of object files:

- From the Build menu, choose the Build command.
- In the Project Configuration Files dialog box, click the Build button. (For details about the Project Configuration Files dialog box, refer to Subsection 2.4.3.2, [4].)
- Click the build start button in the toolbar.
- While holding down the Ctrl key, press the B key.

If building ends normally, the screen shown below will appear.

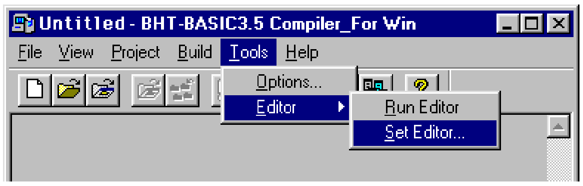

Projtest.bhp[Test1.src] - BHT-BASIC3.5 Compiler_For Win
File View Project Build Tools Help

```
Compiling Test1.src now.
0001 Error statement Compiled End
Compiling finished normally.
Compiling Test2.src now.
0000 Error statement Compiled End
Compiling finished normally.
Compiling Test3.src now.
0000 Error statement Compiled End
Compiling finished normally.
Compiling Test4.src now.
0000 Error statement Compiled End
Compiling finished normally.
Compiling Test5.src now.
0000 Error statement Compiled End
Compiling finished normally.
Linking Protest.bhp now.
0000 Error statement Linked End
Building finished normally.
```

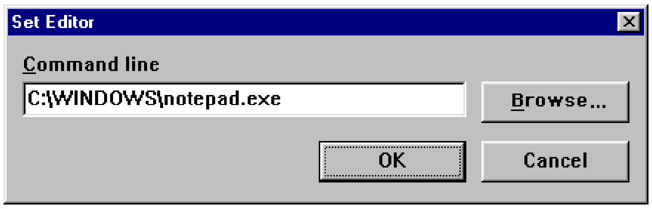
2.4.5 Setting the Editor for Displaying Files

Set the editor that you want to use for displaying source files and error message files (XXX.ERR) according to the steps below.

(1) From the Tools menu, choose the Set Editor command.



The Set Editor dialog box appears as shown below.



C:\WINDOWS\notepad.exe

(2) In the Command line edit box, type the filename of the editor. If the editor is not located in the current directory or working directory, type the absolute path or relative path. (The default editor is Windows NotePad.)

If you don't know the editor's filename or directory path, choose the Browse button in the Set Editor dialog box to display the Select Editor dialog box. From a list of files and directories displayed, select the appropriate filename and then choose the OK button.

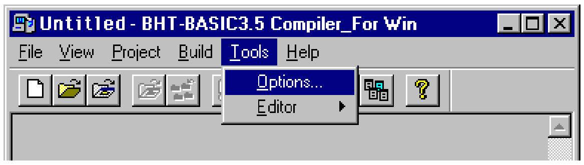
TIP Setting the editor having the tag-jump function allows you to efficiently correct a source program file which has caused an error. For details about the tag-jump function, refer to the user's manual of the editor.

2.4.6 Error Messages and Their Indication onto the Main Window

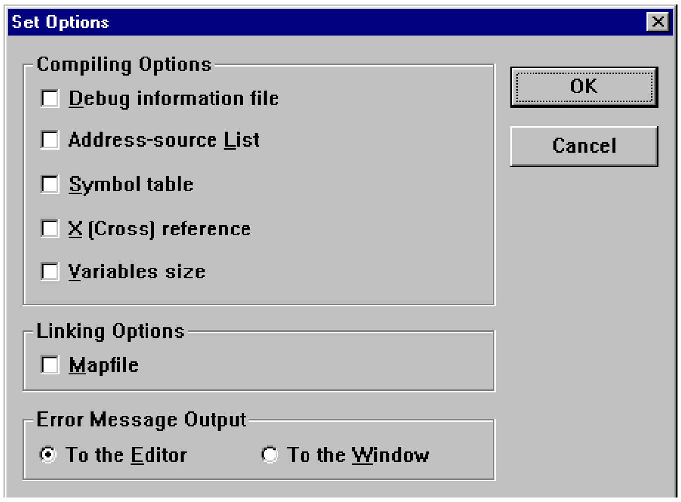
[1] Selecting either an editor or main window as an error message output device

According to the procedure below, you may select whether error messages should be outputted to an editor or main window if an error message file (XXX.ERR) is produced.

(1) From the Tools menu, choose the Options command.


Untitled - BHT-BASIC3.5 Compiler_For Win
File View Project Build Tools Help
Options...
Editor

The Set Options dialog box appears as shown below.


Set Options

Compiling Options
- [ ] Debug information file
- [ ] Address-source List
- [ ] Symbol table
- [ ] X (Cross) reference
- [ ] Variables size

Linking Options
- [ ] Mapfile

Error Message Output
[✓ To the Editor] To the Window

(2) In the Set Options dialog box, select either "To the Editor" or "To the Window" check box. (The default output device is Editor.)

[2]   How error messages are displayed on the editor or main window

During building, the BHT-BASIC 3.5 Compiler may detect errors which can be divided into two types: syntax errors and fatal errors.

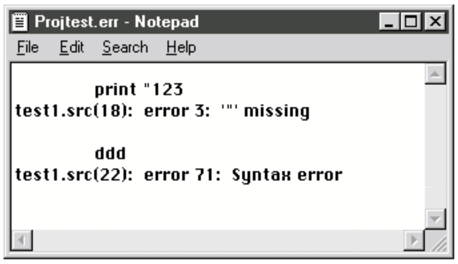
■ Syntax errors

If the Compiler detects a syntax error, it outputs the error message to the XXX.ERR file. For details about the file, refer to Subsection 2.4.9, "Output from the BHT-BASIC 3.5 Compiler."

If the "To the Editor" check box of the Error Message Output is selected in the Set Options dialog box, the editor will automatically open and show the detected errors. If the "To the Window" check box is selected, those errors will be outputted to the main window.

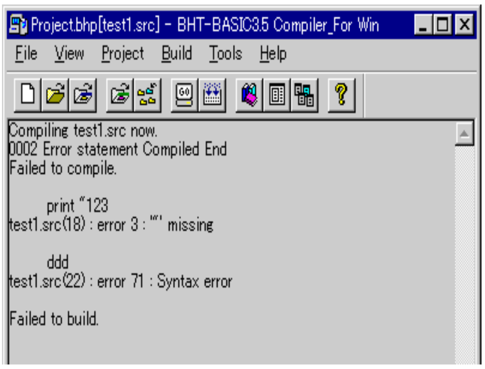
The total number of detected syntax errors always displays on the main window.

- Error messages displayed on the editor



```
          print "123
test1.src(18): error 3: '"' missing

          ddd
test1.src(22): error 71: Syntax error
```

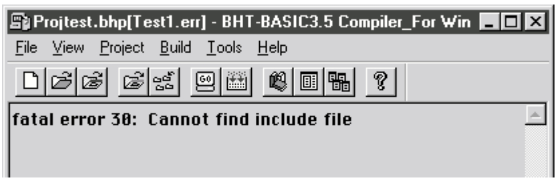
- Error messages displayed on the main window


Project.bhp[test1.src] - BHT-BASIC3.5 Compiler_For Win
File View Project Build Tools Help

```
Compiling test1.src now.
0002 Error statement Compiled End
Failed to compile.

     print "123
test1.src(18) : error 3 : '"' missing

     ddd
test1.src(22) : error 71 : Syntax error

Failed to build.
```

■ Fatal errors

If the Compiler detects a fatal error, it outputs the error message to the main window.


Projtest.bhp[Test1.err] - BHT-BASIC3.5 Compiler_For Win
File View Project Build Tools Help

```
fatal error 30: Cannot find include file
```

■ ERRORLEVEL

The ERRORLEVEL function is supported only when a +E option is specified at the command line. (Refer to Subsection 2.4.8, "Starting the BHT-BASIC 3.5 Compiler from the Command Line," [3].)

2.4.7 Options

To specify compiling options and linking options, select the check-box options you want in the Set Options dialog box. Each of available options is explained below.

[1] Compiling options

Compiling Options	Description
Debug information file	Outputs debug information files (XXX.ADR, XXX.LBL, and XXX.SYM files). If this option is not selected, no debug information file will be outputted. (default) (For details, refer to [3].)
Address-source List	Outputs an address-source list to the file XXX.LST. If this option is not selected, no address-source list will be outputted. (default) (For details, refer to [4].)
Symbol table	Outputs a symbol table to the file XXX.LST. If this option is not selected, no symbol table will be outputted. (default) (For details, refer to [4].)
X (Cross) reference	Outputs a cross reference to the file XXX.LST. If this option is not selected, no cross reference will be outputted. (default) (For details, refer to [4].)
Variable size	Outputs the sizes of common variables, work variables, and register variables to the file XXX.ERR. or main window. If this option is not selected, no variable size will be outputted. (default) The output example (TESTA.err) is as follows: `Common area   = XXXXX bytes (XXXXX bytes on memory.` `XXXXX bytes in file)` `Work area     = XXXXX bytes (XXXXX bytes on memory.` `XXXXX bytes in file)` `Register area = XXXXX bytes in file`

[2] Linking options

Linking Options	Description
Mapfile	Outputs map information to the file XXX.MAP. If this option is not selected, no map information will be outputted. (default) (For details, refer to [5] in this subsection.)

[3] Outputting debug information files

If you select the "Debug information file" check box in the Set Options dialog box and run the Compiler, then the Compiler will output three types of debug information files.

Each information file will be given the same name as the source program and annexed one of the three extensions .ADR, .LBL, and .SYM according to the file type as listed below.

Debug Information Files	Filename Extension
Source line–address file	.ADR
Label-address file	.LBL
Variable–intermediate language file	.SYM

• Source line–address file (.ADR)

Indicates the correspondence of line numbers in a source program to their addresses in the object program written in intermediate language.

Each line consists of a four-digit line number in decimal notation and a four-digit address in hexadecimal notation.

• Label–address file (.LBL)

Indicates the correspondence of labels and user-created functions defined in a source program to their addresses in the object program written in intermediate language.

For user-defined functions in the one-line format, the first addresses of those functions in the object program are listed in this file; for those in the block format, the addresses of the first statements in the blocks are listed.

Each line consists of a label name or a user-defined function name, and a four-digit address in hexadecimal notation.

• Variable–intermediate language file (.SYM)

Indicates the correspondence of variables used in a source program to the intermediate language.

Each line consists of a variable and its intermediate language.

[4] Outputting list files

The Compiler may output three types of list files as listed below depending upon the options specified at the start of compiling, in order to help you program and debug efficiently.

List File	Option	Filename Extension
Address-source list	Select the Address-source List check box.	
Symbol table	Select the Symbol table check box.	.LST
Cross reference	Select the X (Cross) reference check box.	

The list file will be given the same name as the source program file and annexed with an extension .LST.

When outputted, each list file has the header format as shown below.

```
BHT-BASIC 3.5 Compiler Version X.XX ←Version of BHT35CPL.DLL
Copyright (C) DENSO CORPORATION 1998. All rights reserved.
source = Source filename.ext (to be given as an absolute path)
```

■ Address-source list

Select the Address-source List check box and run the Compiler, and the following information will be outputted:

```
BHT-BASIC 3.5 Compiler Version X.XX
Copyright (C) DENSO CORPORATION 1998. All rights reserved.
source = C:\TEST.SRC

Addr      Line      Statement
                                                           Address of object program in
0000      0001      '* * * * * * * * * * * * * *             intermediate language
0000      0002      '*
0000      0014      ON  ERROR  GOTO  ErrorProg
0003      0015                                             Line number in source
0003      0016                 DEFREG  vF% = 0              program
0003      0017                 DEFREG  ConF% = 0
0003      0018                 DEFREG  RecF% = 0
0003      0019                 DEFREG  FreeSpace
0003      0020                 DEFREG  ESC = -1
0003      0021                 DEFREG  bps$ = "9600"
0003      0022
0338      0023      REM  $ INCLUDE : 'SAKeyFnc. SRC'       Source program statement
0338      0024
0338      0025                 Master$    = "Master92. DAT"
034A      0026                 Workfile$    = "WrkFils. DAT"
035C      0027                 Sales$    = "SalesSA. DAT"
036D      0028
036D      0029                 IF vf%   =  0  THEN
0377      0030                           GOSUB  cautionB
037A      0031                           CLOSE
037E      0032                           Freespace   =  FRE(1)
0387      0033                           vF%   =  1
038E      0034                 END IF
038E      0035      MainProg:
038E      0036                 GOSUB  filOpen
  0000 Error Statement Compiled End.
```

- **Address of object program in intermediate language**

  Shows an intermediate language address corresponding to a source program line in four-digit hexadecimal notation.

- **Line number in source program**

  Shows a line number for a source program statement in four-digit decimal notation.

- **Source program statement**

  Shows the same content as a statement written in a source program.

Notes for address-source lists

(1) If a source program statement contains line feeding caused by a CR code preceded by an underline (_) or a comma (,), the line number will increase, but no address will appear.

(2) Neither page headers nor new page codes will be inserted.

(3) If a syntax error occurs, the error message will be outputted on the line following the error statement line.

(4) If more than one syntax error occurs in a statement, the error message only for the first detected error will appear.

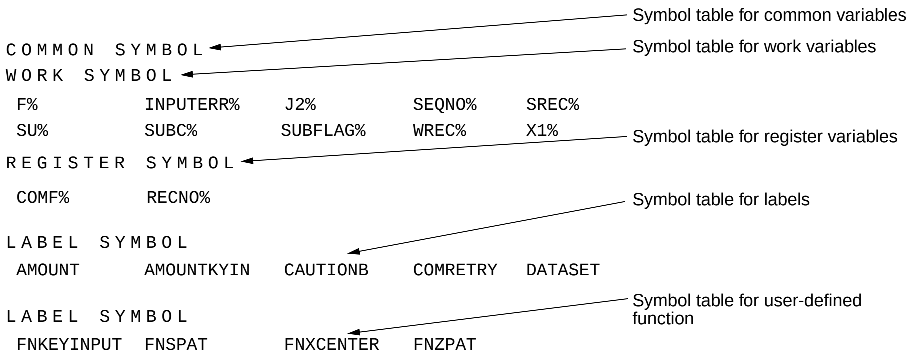
(5) A TAB code will be replaced with eight space codes.

The total number of syntax errors will be outputted at the end of the list.

■ Symbol table

Select the Symbol table check box and run the Compiler, and the following information will be outputted:

```
BHT-BASIC 3.5 Compiler Version X.XX
Copyright (C) DENSO CORPORATION 1998. All rights reserved.
source = C:\Test.SRC
```



```
COMMON SYMBOL
WORK SYMBOL
 F%          INPUTERR%   J2%         SEQNO%     SREC%
 SU%         SUBC%       SUBFLAG%    WREC%      X1%
REGISTER SYMBOL
 COMF%       RECNO%

LABEL SYMBOL
 AMOUNT      AMOUNTKYIN  CAUTIONB    COMRETRY   DATASET

LABEL SYMBOL
 FNKEYINPUT  FNSPAT      FNXCENTER   FNZPAT
```

Symbol table for common variables
Symbol table for work variables
Symbol table for register variables
Symbol table for labels
Symbol table for user-defined function

Variables will be outputted in the following format:

In case of global variables	Variablename
In case of local variables	Variablename:Filename (no extension)-
In other cases	Variablename:Name of user-defined function defining the variable

- **Symbol table for common variables**

  Lists common variables arranged according to their types. An array variable has a suffix of parentheses ( ).

- **Symbol table for work variables**

  Lists work variables and dummy arguments arranged according to their types. An array variable has a suffix of parentheses ( ).

- **Symbol table for register variables**

  Lists register variables arranged according to their types. An array variable has a suffix of parentheses ( ).

- **Symbol table for labels**

  Lists labels arranged in alphabetic order.

- **Symbol table for user-defined functions**

  Lists user-defined functions arranged according to their types (i.e. integer, real, and string types).

Each of common variables, work variables, and register variables can be divided into the following types:

Non-array integer type	Non-array real type	Non-array string type
Array integer type	Array real type	Array string type

■ Cross reference

Select the X (Cross) reference check box and run the Compiler, and the following information will be outputted:

- **For common variables**

  Outputs line numbers where common variables are defined and referred to.

- **For work variables**

  Outputs line numbers where work variables and dummy arguments are referred to.

- **For register variables**

  Outputs line numbers where register variables are defined and referred to.

- **For labels**

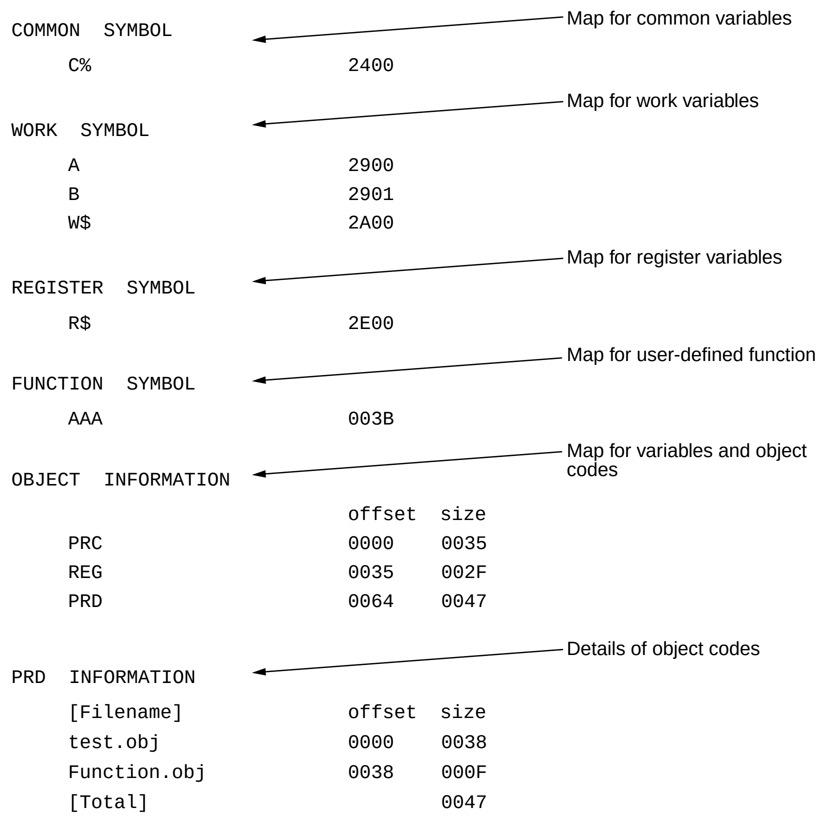
  Outputs line numbers where labels are defined and referred to.

- **For user-defined functions**

  Outputs line numbers where user-defined functions are defined and referred to.

[5] Outputting a mapfile

Select the Mapfile check box of the Linking Options in the Set Options dialog box and build a user program, and the mapfile as shown below will be outputted. The mapfile will be given the same name as the project file and annexed with an extension .MAP.


Map for common variables
Map for work variables
Map for register variables
Map for user-defined function
Map for variables and object codes
Details of object codes

```
COMMON  SYMBOL
     C%                        2400

WORK  SYMBOL
     A                         2900
     B                         2901
     W$                        2A00

REGISTER  SYMBOL
     R$                        2E00

FUNCTION  SYMBOL
     AAA                       003B

OBJECT  INFORMATION
                               offset  size
     PRC                       0000    0035
     REG                       0035    002F
     PRD                       0064    0047

PRD  INFORMATION
     [Filename]                offset  size
     test.obj                  0000    0038
     Function.obj              0038    000F
     [Total]                           0047
```

- **Map for common variables**

  Shows the symbols of common variables in the Interpreter which are arranged according to their types together with their pointing addresses. An array variable has a suffix of parentheses ( ). If no common variables are used, this item will not be outputted.

- **Map for work variables**

  Shows the symbols of work variables in the Interpreter which are arranged according to their types together with their pointing addresses. An array variable has a suffix of parentheses ( ). If no work variables are used, this item will not be outputted.

- **Map for register variables**

  Shows the symbols of register variables in the Interpreter which are arranged according to their types together with their pointing addresses. An array variable has a suffix of parentheses ( ). If no register variables are used, this item will not be outputted.

- **Map for user-defined functions**

  Shows the symbols of user-defined functions in the Interpreter which are arranged according to their types (i.e., integer, real, and string types). If no user-defined functions are used, this item will not be outputted.

- **Map for variables and object codes**

  Shows the addresses of variables and object codes in a user program. The PRC indicates the program allocation information area, the REG indicates the register variables area, and the PRD indicates the program reserved area.

- **Details of object codes**

  Shows the allocation information of objects in a user program. The [Filename] lists the names of object files configuring a user program. The [Offset] lists the heading addresses of individual object files in 4-digit hexadecimal form. The [Size] lists the sizes of individual object files in 4-digit hexadecimal form.

[6] Calculating the address for a statement causing a run-time error

If a run-time error occurs, the Compiler returns the address (ERL=XXXX) assigned starting from the head of the user program. When building a user program out of multiple object files, therefore, you need to calculate an address of a statement in an object file causing a run-time error according to the procedure given below.

(1) In the Set Options dialog box, select the Address-source List check box of the Compiling Options and the Mapfile check box of the Linking Options beforehand.

(2) Build a user program out of object files so as to output the address-source list file (source filename.LST) and the mapfile (project name.MAP).

(3) In the "details of object codes" item, retrieve an object file containing the address (ERL=XXXX) assigned to a statement causing a run-time error.

(4) In the Address-source List file of the retrieved object file, retrieve the address for the statement causing a run-time error.

    Subtract the heading address of the object file from the address of the statementstatement causing a run-time error, and you can obtain where a run-time error has occurred.

2.4.8 Starting the BHT-BASIC Compiler from the Command Line

You may start the BHT-BASIC Compiler from the command line in the MS-DOS Prompt of Windows95/98 or WindowsNT4.0/Windows2000.

[1] Syntax

At the MS-DOS command prompt, type in the following format:

```
BHTC35W [options] [[directorypath]filename…][options]
```

directorypath	You may specify either an absolute path or relative path. Omitting this option will make the Compiler look for that file in the current working directory. Specifications of directorypath only is not allowed.
filename	You may specify the name of any of a project file, source file and library file.
options	You may specify compiler processing options, compiling options, and linking option. For details, refer to the next item, [2], "Options."

NOTE The Compiler will recognize a project specified by filename merely as a group of files. If you do not specify a +BL option (Building library described in [2]), therefore, the Compiler automatically produces a user program.

TIP To produce a user program from a single source file in a batch file, type in the following:

```
>START /W +E +B TEST.SRC
```

Writing START /W as above will not proceed to the next batch processing until the BHT-BASIC 3.5 Compiler completes the processing. For details about +E or +B option, refer to “[2] Options” in this subsection.

[2] Options

The BHT-BASIC 3.5 Compiler supports three types of options—compiler processing options, compiling options, and linking option.

■ Compiler processing options

Processing options	Description
`+C`	Compiles one or more designated file(s) into object file(s).
`+B programname`	Builds a user program with the specified program name. If no `programname` is specified, the filename specified first will apply.
`+BL libraryname`	Builds a library with the specified library name. If no `libraryname` is specified, the filename specified first will apply.
`+E, -E`	Determines whether to terminate the BHT-BASIC 3.5 Compiler after completion of processing. Specifying the `+E` terminates the Compiler without displaying the compiler window after completion of processing. Specifying the `-E` displays the compiler window and does not terminate the Compiler even after completion of processing. The default is `-E`.

NOTE If more than one option with different specifications is written (e.g., +C, +B, and +BL), the last option takes effect.

If the same option is set more than one time with different specifications (e.g., +E and -E), the last option takes effect.

■ Compiling options

Compiling options	Description
`+D`	Outputs debug information files (XXX.ADR, XXX.LBL. and XXX.SYM files). (Same as you select the Debug information file check box in the Set Options dialog box. Refer to Subsection 2.4.7, [1].)
`+L`	Outputs an address-source list to the file XXX.LST. (Same as you select the Address-source List check box in the Set Options dialog box. Refer to Subsection 2.4.7, [1].)
`+S`	Outputs a symbol table to the file XXX.LST. (Same as you select the Symbol table check box in the Set Options dialog box. Refer to Subsection 2.4.7, [1].)
`+X`	Outputs a cross reference to the file XXX.LST. (Same as you select the X (Cross) reference check box in the Set Options dialog box. Refer to Subsection 2.4.7, [1].)
`+V`	Outputs the sizes of common variables, work variables, and register variables to the file XXX.ERR or main window. (Same as you select the Variable size check box in the Set Options dialog box. Refer to Subsection 2.4.7, [1].)

■ Linking option

Linking options	Description
`+M`	Outputs map information to the file XXX.MAP. (Same as you select the Mapfile check box in the Set Options dialog box. Refer to Subsection 2.4.7, [2].)

NOTE

Options specified at the command line will take effect only when you run the BHT-BASIC 3.5 Compiler at the command line. (Those option settings will not be written into the initialization file BHTC35W.INI.)

Even if you specify a -E option (default) so that the Compiler does not terminate after completion of processing, neither filename nor options designated for the preceding processing will be saved. You need to designate them again.

Option settings stored in the initialization file BHTC35W.INI will not apply when you run the BHT-BASIC 3.5 Compiler at the command line. To output debug information files, therefore, you need to specify options at the command line.

[3] Error Level Indication by ERRORLEVEL

If you specify a +E option at the command line and run the BHT-BASIC 3.5 Compiler, the ERRORLEVEL of MS-DOS allows the Compiler to set the compiling end status to the MS-DOS environmental variable ERRORLEVEL after completion of processing, as any of the error levels listed below.

By referring to this ERRORLEVEL, you can learn the compiling end status.

ERRORLEVEL	Description
0	Normal end
1	No designated file or path found.
2	Filename format not correct
4	Project invalid
5	File open error
6	Write-protect error
7	File renaming failure
8	Project file creating failure
9	Existing project file deleted
10	Entered option invalid
20	Compiling syntax error
21	Compiling fatal error
30	Link error
40	Library error
70	No empty space in the designated disk
99	Other errors

By making a batch file which automatically starts proper operation according to the error level, you can facilitate debugging procedures.

For details about the ERRORLEVEL, refer to the MS-DOS Reference Manual.

2.4.9 Output from the BHT-BASIC 3.5 Compiler

The BHT-BASIC 3.5 Compiler outputs the following information as well as object programs to the destination depending upon the conditions.

Output		Destination	Conditions
Object file		File XXX.OBJ (in the directory where the source program is located)	When the specified source program has been normally compiled without occurrence of a compiling error.
User program		File YYY.PD3 (in the directory where the project is located)	When the specified project has been normally built without occurrence of a compiling error or linking error.
Library file		File YYY.LIB (in the directory where the project is located)	When the specified project has been normally built without occurrence of a compiling error or library error.
Error message (Syntax error)		File XXX.ERR (in the directory where the source program is located)	If a compiling error is detected during compilation of the specified source program.
		File YYY.ERR (in the directory where the project is located)	If an error is detected during building of the specified project.
Error message (Fatal error)		Main window	If a fatal error is detected during compilation of the specified source program.
Debug information	Source line–Address information	File XXX.ADR (in the directory where the source program is located)	If the Debug information file check box is selected in the Set Options dialog box.
	Label–Address information	File XXX.LBL (in the directory where the source program is located)	
	Variable–Intermediate language information	File XXX.SYM (in the directory where the source program is located)	

Output	Destination	Conditions
Address–Source list	File XXX.LST (in the directory where the source program is located)	If the Address-source List check box is selected in the Set Options dialog box.
Symbol table		If the Symbol table check box is selected in the Set Options dialog box.
Cross reference		If the X (Cross) reference check box is selected in the Set Options dialog box.
Sizes of variables	File XXX.ERR (in the directory where the source program is located) or File YYY.ERR (in the directory where the project is located)	If the Variable size check box is selected in the Set Options dialog box.
Mapfile	File YYY.MAP (in the directory where the project is located)	If the Mapfile check box is selected in the Set Options dialog box.

XXX represents a source program filename.

YYY represents a project name.

2.4.10 Structure of User Programs and Libraries

If you specify a user program to be produced in the Project Configuration Files dialog box, the BHT-BASIC 3.5 Compiler produces a user program provided that no compiling error or link error occurs. The user program file will be given the same name as the project file and annexed with an extension .PD3.

If you specify a library to be produced, the Compiler produces a library provided that no compiling error or library error occurs. The library file will be given the same name as the project file and annexed with an extension .LIB.

If the name of a newly produced file is the same as that of an existing file in the destination directory, Compiler will overwrite the existing file with the new file.

Structure of user programs

A user program is expressed in the intermediate language, where statements, functions and variables are in two-byte form of ASCII characters. A record is 128 bytes in length and annexed with CR and LF codes.

When downloaded to the BHT and stored in its memory, a user program will be compressed from two-byte form into single-byte hexadecimal form. Accordingly, the length of a record comes to 64 bytes.

Structure of libraries

A library consists of more than one object filename and object information.

2.5 Downloading

2.5.1 Overview of Transfer Utility/Ir-Transfer Utility C/ Ir-Transfer Utility E

Transfer Utility/Ir-Transfer Utility C/Ir-Transfer Utility E transfers user programs and data files (e.g., master files) between the BHT and the connected personal computer. It has the following functions:

Functions of Transfer Utility/Ir-Transfer Utility C/Ir-Transfer Utility E
Downloading extension programs
Downloading programs
Downloading data
Uploading programs
Uploading data

For operations of Transfer Utility/Ir-Transfer Utility C/Ir-Transfer Utility E, refer to the related guide.

2.5.2 Setting up the BHT

■ BHT-3000

If the error message "Report to the personnel in charge (x)" appears, it is necessary to initialize the BHT before downloading user programs.

The above error message appears in any of the following cases:

- The BHT is first powered on from the time of purchase.
- The BHT is powered on after being left without batteries (main and sub) loaded for a long time.

CAUTION Initialization will not only erase all of the programs and data stored in the RAM but also reset the system calendar clock and communications parameters to their defaults. Therefore, set those reset parameters in System Mode before accessing the download menu.

For details about the initialization and downloading, refer to the "BHT-3000 User's Manual."

■ BHT-4000

If the error message "No System" appears, it is necessary to download the System Programs and initialize the BHT before downloading user programs. If the error message "Report to the personnel in charge!!" appears, it is necessary to initialize the BHT.

The error message "No System" appears in any of the following cases:

- The BHT is first powered on from the time of purchase.
- The BHT is powered on after being left without main battery loaded for a long time.

CAUTION Initialization will not only erase all of the programs (including Easy Pack) and data stored in the user area of the RAM but also reset the system calendar clock and communications parameters to their defaults. Therefore, set those reset parameters in System Mode before accessing the download menu.

For details about the initialization and downloading (of System Program and user programs), refer to the "BHT-4000 User's Manual."

■ BHT-5000/BHT-6000/BHT-6500

If the error message given below appears, it is necessary to initialize the BHT before downloading user programs.

"System error ! Contact your administrator. Note the error drive. (DRIVE x)"

The above error message appears in any of the following cases:

- The BHT is first powered on from the time of purchase.
- The BHT is powered on after being left without main battery loaded for a long time.

CAUTION Initialization will not only erase all of the programs and data stored in the RAM and flash ROM but also reset the system calendar clock and communications parameters to their defaults. Therefore, set those reset parameters in System Mode before accessing the download menu.

For details about the initialization and downloading, refer to the "BHT's User's Manual."

■ BHT-7000/BHT-7500

If the error message given below appears, it is necessary to set the calendar clock before downloading user programs.

"Set the current date and time. XX/XX/XX YY:YY"

The above error message appears in any of the following cases:

- The BHT is first powered on from the time of purchase.
- The BHT is powered on after being left without main battery loaded for a long time.

For details about the calendar clock setting, refer to the "BHT's User's Manual."

2.6 Executing a User Program

2.6.1 Starting

To run a user program, start System Mode and select the desired program in the Program Execution menu.

If you have selected a user program as an execution program in the Setting menu of System Mode, the BHT automatically runs the user program when powered on.

For the operating procedure of System Mode, refer to the BHT's User's Manual.

2.6.2 Execution

The Interpreter interprets and executes a user program from the first statement to the next, one by one.

2.6.3 Termination

The BHT system program terminates a running user program if

- the `END`, `POWER OFF`, or `POWER 0` statement is executed in a user program,
- the power switch is pressed,
- no valid operations are performed within the specified time length (Automatic powering-off), or

Valid operations:	- Entry by pressing any key
	- Bar-code reading by pressing the trigger switch
	- Data transmission
	- Data reception
Specified time length:	Length of time specified by the POWER statement in the user program. If not specified in the program, three minutes will apply.

- the battery voltage level becomes low.

Low battery:	If the voltage level of the battery cartridge or that of the dry cells drops below the specified level, the BHT displays the low battery warning message on the LCD and powers itself off.

If the resume function is activated in System Mode, only the execution of the `END`, `POWER OFF`, or `POWER 0` statement can terminate a running user program. Other cases above merely turn off the power, so turning it on again resumes the program.

Chapter 3
Program Structure

CONTENTS

3.1 Program Overview

3.1.1 Statement Blocks

A statement block is a significant set of statements (which is also called "program routine"). The following types of statement blocks are available in programming for the BHT:

Statement Blocks	Description
Subroutine	A routine called by the GOSUB statement.
Error-/event-handling routine	An error-/event-handling routine to which control is passed when an error trap or event (of keystroke) trap occurs, respectively.
User-defined function	A function defined by any of the following statements: DEF FN (in single-line form) DEF FN...END DEF (in block form) SUB...END SUB FUNCTION...END FUNCTION

Block-structured statement	FOR...NEXT
	IF...THEN...ELSE...END IF
	SELECT...CASE...END SELECT
	WHILE...WEND

Avoid jumping into or out of the midst of any of the above statement blocks using the GOTO statement; otherwise, it will result in an error. (Refer to Section 3.1.2.)

[1] Subroutines

A subroutine is a statement block called from the main routine or other subroutines by the GOSUB statement.

Using the RETURN statement passes control from the called subroutine back to the statement immediately following the GOSUB statement in the original main routine or subroutine.

[2] Error-/Event-handling Routines

An error- or event-handling routine is a statement block to which program control passes when an error trap or event (of keystroke) trap occurs during program execution, respectively.

The RESUME statement passes control from the error-handling routine back to the desired statement.

The RETURN statement in the keyboard interrupt event-handling routine returns control to the statement following the one that caused the interrupt.

[3] User-defined Functions

Before calling user-defined functions, it is necessary to define those functions with any of the following statements. Generally, those statements should be placed before the main routine starts.

```
        DEF FN  (in single-line form)
        DEF FN...END DEF  (in block form)
        SUB...END SUB
        FUNCTION...END FUNCTION
```

When using SUB and FUNCTION functions written in other files, it is necessary to declare them with the DECLARE statement before calling them.

[4] Block-structured Statements

The statements listed below have the statement block structure and are useful for structured programming.

```
        FOR...NEXT
        IF...THEN...ELSE...END IF
        SELECT...CASE...END SELECT
        WHILE...WEND
```

■ Nested Structure

Block-structured statements allow you to write nesting programs as shown below.

```
        FOR i=1 TO 10
            FOR j=2 TO 10 STEP 2
                PRINT i, j, k
            NEXT j
        NEXT i
```

Nesting subroutines as shown below is also possible.

```
            GOSUB aaa
               .
               .
               .

        aaa
            PRINT "aaa"
            GOSUB bbb
            RETURN
        bbb
            PRINT "bbb"
            RETURN
```

3.1.2 Notes for Jumping into/out of Statement Blocks

It is not recommended to jump control from a main routine or subroutines into the midst of significant statement blocks or to jump out from the midst of those statement blocks, using the `GOTO` statement.

Statement Blocks	Jump into	Jump out
Subroutine	✕	✕
Error-/event-handling routine	✕	✕
Block-format user-defined function	✕	✕
Block-structured statement	✕	▲

✕ : To be avoided. A run-time error may occur.

▲: Not recommended, although no run-time error will result directly. Nesting may cause a run-time error.

- It is possible to jump control out of the midst of block-structured statements (except for `FOR`...`NEXT`) by using the `GOTO` statement.
- Avoid jumping the control out of the midst of `FOR`...`NEXT` statement block with the `GOTO` statement. The program given below, for example, should be avoided.

```
    FOR I%=0 TO 10
        IF I%=5 THEN
            GOTO AAA
        ENDIF
    NEXT I%
AAA
```

NOTE Generally, the frequent or improper use of `GOTO` statements in a program will decrease debugging efficiency and might cause fatal run-time errors. You are, therefore, recommended to avoid using `GOTO` statements, if possible.

3.2 Handling User Programs

3.2.1 User Programs in the Memory

The user area of the memory (memories) in the BHT can store more than one user program. (For details about memories, refer to Appendix F, "Memory Area.")

If you have selected one of those programs as an execution program in the Setting menu of System Mode, the BHT automatically runs the user program when powered on.

For the operating procedure of System Mode, refer to the BHT's User's Manual.

3.2.2 Program Chaining

Program chaining, which is caused by the `CHAIN` statement as shown below, terminates a currently running user program and transfers control to another program.

```
CHAIN "another.PD3"
```

To transfer the variables and their values used in the currently running user program to the chained-to program along the program chain, use the `COMMON` statement as follows:

```
COMMON a$(2),b,c%(3)
CHAIN "another.PD3"
```

The Interpreter writes these declared variable values into the "common variable area" in the memory. To make the chained-to program refer to these values, use the `COMMON` statement again.

```
COMMON a$(2),b,c%(3)
```

In BHT-BASIC, all of the name, type, definition order, and number of `COMMON`-declared variables used in the currently running program should be identical with those in the next program (the chained-to program).

When compiling and linking more than one file to produce a user program, define all necessary common variables in the main object (to be executed first). In other objects, declare common variables required only in that object. If you link an object where common variables not defined in the main object are newly defined, an error will result.

```
' prog1.PD3
COMMON a(10),b$(3),c%
       ⋮
CHAIN "prog2.PD3"
' prog2.PD3
COMMON a(10),b$(3),c%
       ⋮
```

Since the `COMMON` statement is a declarative statement, no matter where it is placed in a source program, the source program will result in the same output (same object program), if compiled.

3.2.3 Included Files

"Included files" are separate source programs which may be called by the `INCLUDE` metacommand.

Upon encounter with the `INCLUDE` metacommand in a source program, the Compiler fetches the designated included file and then compiles the main source program while integrating that included file to generate a user program.

You should specify the name of an included file by using the `REM $INCLUDE` or `'$INCLUDE`. In the included files, you can describe any of the statements and functions except the `REM $INCLUDE` and `'$INCLUDE`.

Storing definitions of variables, subroutines, user-defined functions, and other data to be shared by source programs into the included files will promote application of valuable program resources.

If a compilation error occurs in an included file, it will be merely indicated on the line where the included file is called by the `INCLUDE` metacommand in the main source program, and neither detailed information of syntax errors detected in the included files nor the cross reference list will be outputted. It is, therefore, necessary to debug the individual included files carefully beforehand.

Chapter 4
Basic Program Elements

CONTENTS

4.1 Structure of a Program Line

4.1.1 Format of a Program Line

A program line consists of the following elements:

`[`*`label`*`] [`*`statement`*`] [:`*`statement`*`] ... [`*`comment`*`]`

- **label**

  A label is placed at the beginning of a program line to identify lines.

- **statement**

  A statement is a combination of functions, variables, and operators according to the syntax.

  A group of the statements is a program.

- **comment**

  You may describe comments in order to make programs easy to understand.

[1] Labels

To transfer control to any other processing flow like program branching, you may use labels which designate jump destinations. Labels can be omitted if unnecessary.

Labels differ from line numbers used in the general BASIC languages; that is, labels do not determine the execution order of statements.

You should write a label beginning in the 1st column of a program line. To write a statement following a label, it is necessary to place one or more separators (spaces or tabs) between the label and the statement.

As shown below, using a label in the `IF` statement block can eliminate the `GOTO` statement which should usually precede a jump-destination label.

```
IF a = 1 THEN Check
ELSE 500
ENDIF
```

Where the words "Check" and "500" are used as labels.

For detailed information about labels, refer to Section 4.3.

[2] Statements

Statements can come in two types: executable and declarative statements.

- **Executable statements**

  They make the Interpreter process programs by instructing the operation to be executed.

- **Declarative statements**

  They manage the memory allocation for variables and handle comments. Declarative statements available in BHT-BASIC are listed below.

  ```
  REM  or single quotation mark (’)
  DATA
  COMMON
  DEFREG
  ```

Multi-statements: You can describe multiple statements in one program line by separating them with a colon (:).

[3] Comments

A single quotation mark (’) or REM can begin a comment.

- **Single quotation mark (')**

  A single quotation mark or apostrophe (’) can begin in the first column of a program line to describe a comment.

  When following any other statement, a comment starting with a single quotation mark requires no preceding colon (:) as a delimiter.

  ```
  ’   comment
  PRINT "abc"    ’comment
  ```

- **REM**

  The REM cannot begin in the first column of a program line.

  When following any other statement, a comment starting with a REM requires a preceding colon (:).

  ```
  REM comment
  PRINT "abc"    :REM comment
  ```

4.1.2 Program Line Length

A program line is terminated with a CR code by pressing the carriage return key.

The allowable line length is basically 512 characters excluding a CR code placed at the end of the line.

In either of the following two description ways, however, you can write a program line of up to 8192 characters:

In the samples below, symbol "↓" denotes a CR code entered by the carriage return key.

- Extend a program line with an underline (_) and a CR code.

```
IF (a$ = "," OR a$ = ".") AND b<c _↓
AND EOF(d) THEN ...
```

- Extend a program line with a comma (,) and a CR code.

```
FIELD #1,13 as p$,5 as k$,↓
10 as t$↓
```

Note that the latter description way above (using a comma and CR code) cannot be used for the `PRINT`, `PRINT#`, and `PRINT USING` statements. Only the former way should apply to them.

4.2 Usable Characters

4.2.1 Usable Characters

Listed below are characters which can be used for writing programs. Note that a double quote (") cannot be used inside a character string. Symbols | and ~ inside a character string will appear as ↓ and → on the LCD of the BHT, respectively.

If used outside of a character string, symbols and control codes below have special meaning described in Subsection 4.2.2.

• Alphabet letters	Including both the uppercase and lowercase letters (A to Z and a to z).
• Numerals	Including 0 to 9 for decimal notation, and 0 to 9 and A to F (a to f) for hexadecimal notation.
• Symbols	Including the following: $ % * + – . / < = > " & ' ( ) : ; [ ] { } # ! ? @ \ | ~ , _
• Control codes	CR, space, and tab
• Katakana	e.g., ｱ, ｲ, ｳ～ｿ
• Kanji (2-byte codes) (Full-width characters)	e.g., 漢, ゼ, ア, Ａ, ...
• Kanji (2-byte codes) (Half-width characters)	e.g., A, 1, ｱ, ...

■ Distinction between Uppercase and Lowercase Letters

The Compiler makes no distinction between the uppercase and lowercase letters, except for those used in a character string data. All of the statements below, for example, produce the same effect.

```
PRINT a
print a
PRINT A
print A
```

When used in a character string data, uppercase and lowercase letters will be distinguished from each other. Each of the statements below, for example, produces different display output.

```
PRINT "abc"
PRINT "ABC"
```

4.2.2 Special Symbols and Control Codes

Symbols and control codes used outside of a character string have the following special meaning:

Symbols and control codes	Typical use
$ (Dollar sign)	String suffix for variables or user-defined functions
% (Percent sign)	Integer suffix for variables, constants (in decimal notation), or user-defined functions
* (Asterisk)	Multiplication operator
+ (Plus sign)	• Addition operator or unary positive sign • Concatenation operator in string operation • Format control character in `PRINT USING` statement
– (Minus sign)	Subtraction operator or unary negative sign
. (Period)	• Decimal point • Format control character in `PRINT USING` statement
/ (Slant)	• Division operator • Separator for date information in `DATE$` function
< (Less-than sign)	Relational operator
= (Equal sign)	• Relational operator • Assignment operator in arithmetic or string operation • User-defined function definition expressions in single-line form `DEF FN` • Register variable definition expressions
> (Greater-than sign)	Relational operator
" (Double quote)	A pair of double quotes delimits a string constant or a device file name.
& (Ampersand)	• Integer prefix for constants (in hexadecimal notation), which should be followed by an H. • Format control character in `PRINT USING` statement
' (single quotation mark or apostrophes)	• Initiates a comment. • A pair of apostrophes (single quotations) delimits an included file name.
(Left and right parentheses)	• Delimit an array subscript or a function parameter. • Force the order of evaluation in mathematical, relational, string, and logical expressions.

Symbols and control codes	Typical use
: (Colon)	• Separates statements. • Separates time information in `TIME$` function.
; (Semicolon)	Line feed control character in `INPUT` and other statements.
[ ] (Square brackets)	• Define the length of a string variable. • Define the string length of the returned value of a string user-defined function.
{ } (Braces)	Define the initial value for an array element.
# (Pound sign)	• File number prefix in `OPEN`, `CLFILE`, `FIELD`, and other statements. • Format control character in `PRINT USING` statement
! (Exclamation mark)	Format control character in `PRINT USING` statement
@	Format control character in `PRINT USING` statement
' (Comma)	• Separates parameters or arguments. • Line feed control character in `INPUT` and other statements.
_ (Underline)	If followed by a CR code, an underline extends one program line up to 8192 characters.
CR code (Enter)	Terminates a program line.
(Half-width space)	Separator which separates program elements in a program line. (Note that a two-byte full-width space cannot be used as a separator.)
TAB (Tab code)	Separator which separates program elements in a program line.

4.3 Labels

A label can contain the following characters:

- Alphabet characters
- Numeral characters
- Period (.)

■ Rules for naming labels

- The label length should be limited to 10 characters including periods.
- A program can contain up to 9999 labels.
- Label names make no distinction between uppercase and lowercase letters.

  The following labels, for example, will be treated as the same label.

  ```
  filewrite
  FILEWRITE
  FileWrite
  ```

- No asterisk (*) or dollar sign ($) should be used for a label. The following label examples are invalid:

  ```
  *Label0
  Label1$
  ```

- A label made up of only numeral letters as shown below is valid.

  ```
  1000
  1230
  ```

  Note that a single 0 (zero) should not be used as a label name since it has a special meaning in `ON ERROR GOTO, ON KEY...GOSUB`, and `RESUME` statements.

- A reserved word cannot be used by itself for a label name, but can be included within a label name as shown below.

  ```
  inputkey
  ```

- A label should not start with the character string `FN`.

4.4 Identifiers

Identifiers for the names of variables should comprise the same alphanumerics as the labels.

■ Rules for naming identifiers

- The identifier length should be limited to 10 characters including periods and excluding $ (dollar sign) and % (percent sign) suffixes.
- Every type of variables can contain up to 255 identifiers.
- A reserved word cannot be used by itself for an identifier name, but can be included within an identifier name.
- An identifier should not start with a numeral character or the character string FN. If starting with an FN, the character string will be treated as a function identifier defined by the DEF FN statement.

Examples of identifiers:

```
a
abcdef$
a1
a12345%
```

4.5 Reserved Words

"Reserved words" are keywords to be used in statements, functions, and operators. For the reserved words, refer to Appendix B, "Reserved Words."

■ Rules for using reserved words

- A reserved word cannot be used by itself for a label name, a variable name, or other identifiers, but can be included within them. The following identifiers, for example, are improper since they use reserved words "input" and "key" as is, without modification:

```
input = 3
key = 1
```

- A reserved word can be used for a data file name as shown below.

```
OPEN "input" AS #1
```

Chapter 5
Data Types

CONTENTS

5.1 Constants

5.1.1 Types of Constants

A constant is a data item whose value does not change during program execution. Constants are classified into two types: string constants and numeric constants.

Constant			**Example**
String constants			`"ABC", "123"`
Numeric constants	Integer constants	In decimal notation	`123%, -4567`
		In hexadecimal notation	`&HFFF, &h1A2B`
	Real constants		`123.45, -67.8E3`

[1] String Constants

A "string constant" is a character string enclosed with a pair of double quotation marks ("). Its length should be a maximum of 255 characters.

The character string should not contain a double quotation mark (") or any control codes.

[2] Numeric Constants

■ Integer Constants

– In decimal notation

An integer constant in decimals is usually followed by a percent sign (%) as shown below, but the % can be omitted.

Syntax: `sign decimalnumericstring%`

Where the `sign` is either a plus (+) or a minus (–). The plus sign can be omitted.

The valid range is from -32768 to 32767.

If included in an integer constant in decimals, a comma (,) for marking every three digits will cause a syntax error.

– In hexadecimal notation

Integer constants in hexadecimals should be formatted as shown below.

Syntax: `&Hhexnumericstring`

The valid range is from 0h to FFFFh.

If included in a numeric string in hexadecimals, a period denoting a decimal point will cause a syntax error.

■ Real Constants

Real constants should be formatted as shown below.

Syntax: `sign mantissa`

Syntax: `sign mantissa E sign exponent`

Where a lowercase letter "`e`" is also allowed instead of uppercase letter "`E`."

`mantissa` is a numeric string composed of a maximum of 10 significant digits. It can include a decimal point.

If included in a real constant as shown below, a comma (,) for marking every three digits will cause a syntax error.

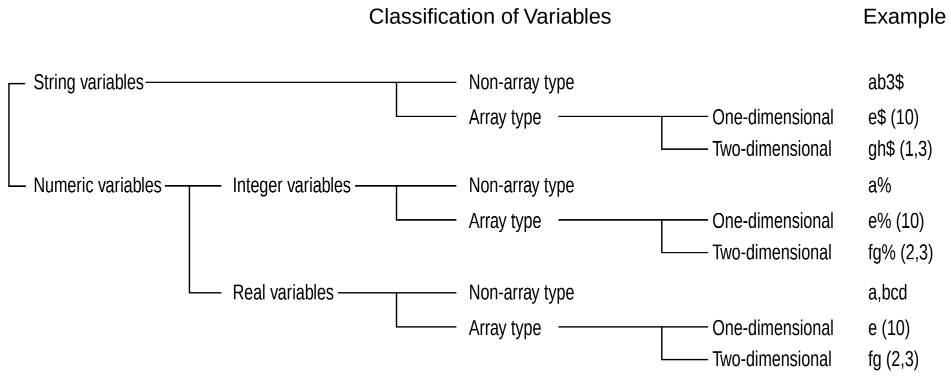
```
123,456        'syntax error!
```

5.2 Variables

A variable is a symbolic name that refers to a unit of data storage. The contents of a variable can change during program execution.

5.2.1 Types of Variables according to Format

Variables are classified into two types: string variables and numeric variables, each of which is subclassified into non-array and array types.



Array variables should be declared in any of the `DIM`, `COMMON`, and `DEFREG` statements. Note that the `DIM` statement should precede statements that will access the array variable.

BHT-BASIC can handle array variables up to two-dimensional.

The subscript range for an array variable is from 0 to 254.

[1] String Variables

A string variable should consist of 1 through 255 characters.

- **Non-array string variables**

  A non-array string variable should be formatted with an identifier followed by a dollar sign ($) as shown below.

  Syntax: *`identifier`*`$`

  Example: `a$,bcd123$`

  The default number of characters for a non-array string variable is 40.

- **Array string variables**

  An array string variable should be formatted with an identifier followed by a dollar sign ($) and a pair of parentheses () as shown below.

  Syntax: *`identifier`*`$(`*`subscript`*`[,`*`subscript`*`])`

  Example: `a$(2),bcd123$(1,3)`

  Where a pair of parentheses indicates an array.

  The default number of characters for an array string variable is 20.

■ Memory Occupation

A string variable occupies the memory space by (the number of characters + one) bytes, where the added one byte is used for the character count. That is, it may occupy 2 to 256 bytes.

If a non-array string variable consisting of 20 characters is declared, for example, it will occupy 21-byte memory space.

[2] Numeric Variables

- **Non-array integer variables**

  A non-array integer variable should be formatted with an identifier followed by a percentage sign (%) as shown below.

  Syntax: `identifier%`

  Example: `a%,bcd%`

- **Array integer variables**

  An array integer variable should be formatted with an identifier followed by a percentage sign (%) and a pair of parentheses () as shown below.

  Syntax: `identifier%(subscript[,subscript])`

  Example: `e%(10),fg%(2,3),h%(i%,j%)`

  Where a pair of parentheses indicates an array.

- **Non-array real variables**

  A non-array real variable should be formatted with an identifier only as shown below.

  Syntax: `identifier`

  Example: `a,bcd`

- **Array real variables**

  An array real variable should be formatted with an identifier followed by a pair of parentheses () as shown below.

  Syntax: `identifier(subscript[,subscript])`

  Example: `e(10),fg(2,3),h(i%,j%)`

  Where a pair of parentheses indicates an array.

■ Memory Occupation

A numeric variable occupies 2 bytes or 6 bytes of the memory space for an integer variable or a real variable, respectively.

5.2.2 Classification of Variables

■ Work Variables

A work variable is intended for general use. You may use it either by declaring with the DIM statement as a non-array variable or without declaration as an array variable. The following examples show work variables:

```
DIM a(10),b%(5),c$(1)
d=100:e%=45
FOR count% = s1% TO s2%
NEXT count%
```

At the start of a user program, the Interpreter initializes all of the work variables to zero (0) or a null character string. At the end of the program, all of these variables will be erased.

Upon execution of the DIM statement declaring an array variable, the Interpreter allocates the memory for the array variable. The declared array variable can be erased by the ERASE statement.

■ Common Variables

A common variable is declared by the COMMON statement. It is used to pass its value to the chained-to programs.

■ Register Variables

A register variable is a unique non-volatile variable supported exclusively by BHT-BASIC. It will retain its value (by battery backup) even after the program has terminated or the BHT has been powered off. Therefore, it should be used to store settings of programs and other values in the memory.

The Interpreter stores register variables in the register variables area of the memory which is different from the work variables area.

Like other variables, register variables are classified into two types: string variables and numeric variables, each of which is subclassified into non-array and array types.

The format of register variables is identical with that of general variables. However, you need to declare register variables including non-array register variables with DEFREG statements.

BHT-BASIC can handle array variables up to two-dimensional.

In the BHT-5000/BHT-6000/BHT-6500, when starting a user program stored in the flash ROM for the first time, the Interpreter copies the register variables into the RAM (so that both the flash ROM and RAM store the register variables). When modifying the register variables, the Interpreter changes those stored in the RAM.

When uploading a program file stored in the flash ROM by using the XFILE statement or System Mode, the BHT-5000/BHT-6000/BHT-6500 uploads the program (except for the register variables in the flash ROM) together with the register variables stored in the RAM.

5.3 User-defined Functions

Out of user-defined functions, the `SUB` and `FUNCTION` functions can be called from other files. The `DEF FN` function can be called only in the file where that function is defined and should start with an `FN`.

The `DEF FN` and `FUNCTION` functions are classified into three types: integer functions, real functions, and character functions, each of which should be defined in the following format:

User-defined Function	Format of `DEF FN`	Format of `FUNCTION`
Integer functions	`FN`	*`functionname`* `%`
Real functions	`FN`	*`functionname`*
Character functions	`FN`	*`functionname`* `$`

■ Setting Character String Length of Returned Values of Character Functions

A character function may return 1 through 255 characters. Note that the default character string length results in the returned value of 40 characters.

If the returned value of the character string length is always less than 40 characters, you can use the stack efficiently by setting the actual required value smaller than the default as the maximum length. This is because the Interpreter positions returned values on the stack during execution of user-defined functions so as to occupy the memory area by the maximum length size. To define a function which results in the returned value of one character, for example, describe as follows:

```
DEF FNshort$(i%)[1]
```

On the other hand, if the returned value is more than 40 characters, it is necessary to set the actually required length. To define a function which results in the returned values of 128 characters, for example, describe as follows:

```
DEF FNlong$(i%)[128]
```

■ Dummy Arguments and Real Arguments

Dummy arguments are used for defining user-defined functions. In the example below, `i%` is a dummy argument.

```
DEF FNfunc%(i%)
    FNfunc%=i%*5
END DEF
```

Real arguments are actually passed to user-defined functions when those functions are called. In the example below, 3 is a real argument.

```
PRINT FNfunc%(3)
```

5.4 Type Conversion

5.4.1 Type Conversion

BHT-BASIC has the type conversion facility which automatically converts a value of one data type into another data type during value assignment to numeric variables and operations; from a real number into an integer number by rounding off, and vice versa, depending upon the conditions.

- The Interpreter automatically converts a value of a real into an integer, in any of the following cases:
  - Assignment of real expressions to integer variables
  - Operands for an arithmetic operator `MOD`
  - Operands for logical operators: `AND`, `OR`, `NOT`, and `XOR`
  - Parameters for functions
  - File numbers

  In the type conversion from real into integer, the allowable value range of resultant integer is limited as shown below. If the resultant integer comes out of the limit, a run-time error.

  ```
  -32768 ≤ resultantintegervalue ≤ +32767
  ```

- In assignments or operations from integer to real, the type-converted real will have higher accuracy:

  Syntax: *`realvariable = integerexpression`*

  In the above case, the Interpreter applies the type conversion to the evaluated resultant of the integer expression before assigning the real value to the real variable.

  Therefore, a in the following program will result in the value of 184.5.

  ```
  a=123%*1.5
  ```

5.4.2 Type Conversion Examples

The following examples show the type conversion from real to integer.

■ Assignment of Real Expressions to Integer Variables

When assigning the value of the real expression (right side) to the integer variable (left side), the Interpreter carries out the type conversion.

Syntax: `integervariable = realexpression`

Example: `b% = 123.45`

Where `b%` will become 123.

■ Operands for an Arithmetic Operator MOD

Before executing the `MOD` operation, the Interpreter converts operands into integers.

Syntax: *realexpression* `MOD` *realexpression*

Example: `10.5 MOD 3.4`

Where the result will become identical with `11 MOD 3`.

■ Operands for Logical Operators AND, OR, NOT, and XOR

Before executing each logical operation, the Interpreter converts operands into integers.

Syntax: `NOT` *realexpression*,
realexpression `{AND|OR|XOR}` *realexpression*

Example: `10.6 AND 12.45`

Where the result will become identical with `11 AND 12`.

■ Parameters for Functions

If parameters `i` and `j` of the functions below are real expressions, for example, the Interpreter converts them into integers before passing them to each function.

```
CHR$(i),HEX$(i),LEFT$(x$,i),MID$(x$,i,j),
RIGHT$(x$,i),...
```

■ File Numbers

The Interpreter also rounds off file numbers to integers.

```
EOF(fileno),LOC(fileno),LOF(fileno),...
```

Chapter 6
Expressions and Operators

CONTENTS

6.1 Overview

An expression is defined as a combination of constants, variables, and other expressions which are connected using operators.

There are two types of expressions--numeric expressions and string expressions.

BHT-BASIC has the following types of operators:

Operators	Description
Arithmetic operator	Performs arithmetic operations.
Relational operator	Compares two values.
Logical operator	Combines multiple tests or Boolean expressions into a single true/false test.
Function operator	Performs the built-in or user-defined functions.
String operator	Concatenates or compares character strings.

6.2 Operator Precedence

When an expression contains more than one operator, BHT-BASIC performs the operations in the standard precedence as shown below.

Precedence

1. **Parentheses ( )**

   The parentheses allow you to override operator precedence; that is, operations enclosed with parentheses are first carried out.

   For improving the readability of an expression, you can use parentheses to separate two operators placed in succession.

2. **Function operations**

3. **Arithmetic operations**

Operations	Arithmetic Operators	Precedence
Negation	_	1
Multiplication and division	* and /	2
Modulo arithmetic	MOD	3
Addition and subtraction	+ and _	4

4. **Relational operations**

   `=, <>, ><, <, >, <=, >=, =<, =>`

5. **Logical operations**

Operations	Logical Operators	Precedence
Logical negation	NOT	1
Logical multiplication	AND	2
Logical addition	OR	3
Exclusive logical addition	XOR	4

6. **String operations**

When more than one operator occurs at the same level of precedence, the BHT-BASIC resolves the expression by proceeding from left to right.

```
a=4+5.0/20*2-1
```

In the above example, the operation order is as follows;

```
5.0/20 (=0.25)
0.25*2 (=0.5)
4+0.5  (=4.5)
4.5-1  (=3.5)
```

6.3 Operators

6.3.1 Arithmetic Operators

Arithmetic operators include a negative sign (-) and operators for multiplication (*), division (/), addition (+), and subtraction (-). They also include modulo operator `MOD`.

Operations	Arithmetic Operators	Precedence	Examples
Negation	`-`	1	`-a`
Multiplication and division	`*` and `/`	2	`a*b, a/b`
Modulo arithmetic	`MOD`	3	`a MOD b`
Addition and subtraction	+ and –	4	`a+b, a-b`

■ Modulo Operation (`MOD`)

The `MOD` operator executes the modulo operation; that is, it divides *`expression 1`* by *`expression 2`* (see the format below) and returns the remainder.

Syntax: *`expression1`* `MOD` *`expression2`*

Where one or more spaces or tab codes should precede and follow the `MOD`.

If these expressions include real values, the `MOD` first rounds them off to integers and then executes the division operation. For example, the `MOD` treats expression `8 MOD 3.4` as `8 MOD 3` so as to return the remainder "2".

■ Overflow and Division by Zero

Arithmetic overflow resulting from an operation or division by zero will cause a run-time error. Such an error may be trapped by error trapping.

6.3.2 Relational Operators

A relational operator compares two values. Depending upon whether the comparison is true or false, the operator returns true (–1) or false (0).

With the operation result, you can control the program flow.

The relational operators include the following:

Relational Operators	Meanings	Examples
=	Equal to	A = B
<> or ><	Not equal to	A <> B
<	Less than	A < B
>	Greater than	A > B
<= or =<	Less than or equal to	A <= B
>= or =>	Greater than or equal to	A >= B

If an expression contains both arithmetic and relational operators, the arithmetic operator has higher precedence than the relational operator.

6.3.3 Logical Operators

A logical operator combines multiple tests and manipulates Boolean operands, then returns the results. It is used, for example, to control the program execution flow or test the value of an `INP` function bitwise, as shown in the sample below.

```
IF d<200 AND f<4 THEN ...
WHILE i>10 OR k<0 ...
IF NOT p THEN ...
barcod% = INP(0) AND &h02
```

Listed below are the four types of logical operators available.

Operations	Logical Operators	Precedence
Negation	`NOT`	1
Logical multiplication	`AND`	2
Logical addition	`OR`	3
Exclusive logical addition	`XOR`	4

One or more spaces or tab codes should precede and follow the `NOT`, `AND`, `OR`, and `XOR` operators.

In the logical expressions (or operands), the logical operator first carries out the type conversion to integers before performing the logical operation. If the resultant integer value is out of the range from -32768 to +32767, a run-time error will occur.

If an expression contains logical operators together with arithmetic and relational operators, the logical operators are given lowest precedence.

[1] The `NOT` operator

The `NOT` operator reverses data bits by evaluating each bit in an expression and setting the resultant bits according to the truth table below.

Syntax: `NOT` *`expression`*

Truth Table for `NOT`

Bit in Expression	Resultant Bit
0	1
1	0

For example, `NOT 0` = -1 (true).

The `NOT` operation for an integer has the returned value of negative 1's complement. The `NOT` X, for instant, is equal to –(X+1).

[2] The `AND` operator

The `AND` operator ANDs the same order bits in two expressions on either side of the operator, then sets 1 to the resultant bit if both of these bits are 1.

Syntax: *`expression1`* `AND` *`expression2`*

Truth Table for `AND`

Bit in *`Expression 1`*	Bit in *`Expression 2`*	Resultant Bit
0	0	0
0	1	0
1	0	0
1	1	1

[3] The `OR` operator

The `OR` operator ORes the same order bits in two expressions on either side of the operator, then sets 1 to the resultant bit if at least one of those bits is 1.

Syntax: *`expression1`* `OR` *`expression2`*

Truth Table for `OR`

Bit in *`Expression 1`*	Bit in *`Expression 2`*	Resultant Bit
0	0	0
0	1	1
1	0	1
1	1	1

[4] The `XOR` operator

The `XOR` operator XORes the same order bits in two expressions on either side of the operator, then sets the resultant bit according to the truth table below.

Syntax: *`expression1`* `XOR` *`expression2`*

Truth Table for `XOR`

Bit in *`Expression 1`*	Bit in *`Expression 2`*	Resultant Bit
0	0	0
0	1	1
1	0	1
1	1	0

6.3.4 Function Operators

The following two types of functions are available in BHT-BASIC, both of which work as function operators:

- **Built-in Functions**

  Already built in BHT-BASIC, e.g., ABS and `INT`.

- **User-defined Functions**

  Defined by using `DEF FN` (in single-line form), `DEF FN`...`END DEF` (in block form), `SUB`...`END SUB`, or `FUNCTION`...`END FUNCTION` statement.

6.3.5 String Operators

A character string operator may concatenate or compare character strings.

Listed below are the types of character string operators available.

Operations	Character String Operators	Examples
Concatenation	+ (Plus sign)	`a$+"."+b$`
Comparison	= (Equal)	`a$=b$`
	`<>, ><` (Not equal)	`a$<>b$, a$><b$`
	`>, <, =<, =>, <=, >=` (Greater or less)	`a$>b$, a$=>b$`

- **Concatenation of Character Strings**

  The process of combining character strings is called concatenation and is executed with the plus sign (+). The example below concatenates the character strings, `a$` and `b$`.

```
a$="Work1" : b$ = "dat"
PRINT a$+"."+b$
```

```
Work1.dat
```

■ Comparison of Character Strings

The relational operators compare two character strings according to character codes assigned to individual characters.

In the example below, the expression `a1$<b1$` returns the value of true so as to output -1.

```
a1$="ABC001"
b1$="ABC002"
PRINT a1$<b1$
```

```
-1
```

Chapter 7
I/O Facilities

CONTENTS

7.1 Output to the LCD Screen

7.1.1 Display Fonts

[1] Fonts available on each BHT

Listed below are the fonts available on each BHT series.

(√: Available)

Screen mode	Font size	BHT–3000	BHT–4000	BHT–5000	BHT–6000	BHT–6500	BHT–7000	BHT–7500	Font type	Dots (W x H)
Single-byte ANK* mode	Standard-size	√	√	√	√	√	√	√	ANK chars	6 x 8
	Small-size				√	√	√	√	ANK chars	6 x 6
Single-byte ANK* mode (Double-width)	Standard-size						√	√	ANK chars	12 x 8
	Small-size						√	√	ANK chars	12 x 6
Two-byte Kanji mode	Standard-size	√	√	√	√	√	√	√	Full-width Half-width	16 x 16 8 x 16
	Small-size				√	√	√	√	Full-width Half-width	12 x 12 6 x 12
Two-byte Kanji mode (Double-width)	Standard-size						√	√	Full-width Half-width	32 x 16 16 x 16
	Small-size						√	√	Full-width Half-width	24 x 12 12 x 12
Condensed two-byte Kanji mode			√	√					Full-width Half-width	12 x 16 6 x 16

*ANK: Alphanumerics and Katakana

The ANK mode displays ANK characters listed in Appendices C1 and C2.

The two-byte Kanji mode displays the following characters:

- Half-width: Katakana and alphanumerics
- Full-width: JIS Levels 1 and 2 Kanji, alphabets and symbols

NOTE Half-width Kanji characters differ from ANK characters in size.

[2] Switching the fonts

You may switch the screen mode and font size by using the statements below.

- SCREEN statement
- OUT statement

To specify the single-byte ANK mode, two-byte Kanji mode, or condensed two-byte Kanji mode, use the SCREEN statement as listed below.

Specifies the single-byte ANK mode	`SCREEN 0`
Specifies the two-byte Kanji mode	`SCREEN 1`
Specifies the condensed two-byte Kanji mode	`SCREEN 2`

To specify the normal- or double-width, use the SCREEN statement as listed below.

Specifies the normal-width	`SCREEN ,0` or `SCREEN ,1`
Specifies the double-width	`SCREEN ,2` or `SCREEN ,3`

To specify the standard- or small-size, use the OUT statement as listed below.

Specifies the standard-size	`OUT &H6080,0`
Specifies the small-size	`OUT &H6080,1`

7.1.2 Number of Characters and Coordinates on the LCD

To locate characters on the coordinates of the LCD screen, use the LOCATE statement. To obtain the current cursor position, use the CSRLIN and POS functions.

[1] BHT-3000

Screen mode	Single-byte ANK mode	Two-byte Kanji mode
Columns x Lines	16 x 4	Full-width: 6 x 2 Half-width: 12 x 2

■ Displaying Kanji Characters

The BHT-3000 has no Kanji font, so it requires "Kanji Utility" to display Kanji characters. The "Kanji Utility" may handle up to 1024 Kanji characters.

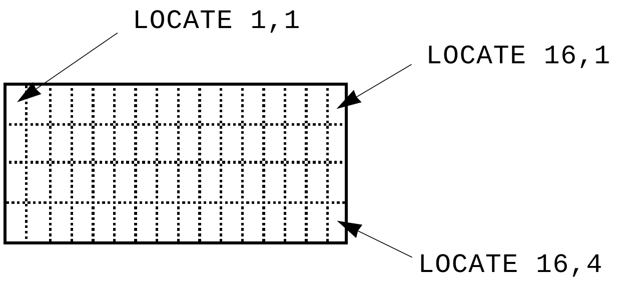
Note that the following characters may be displayed without "Kanji Utility":

- Half-width alphanumerics and Katakana
- Full-width alphanumerics and Katakana
- Full-width Hiragana

■ Locating Characters on the LCD Screen

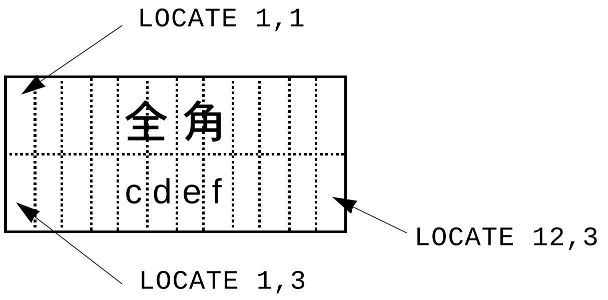
Using the LOCATE statement locates characters on the coordinates of the LCD screen. The coordinates differ depending upon the screen mode as shown below.

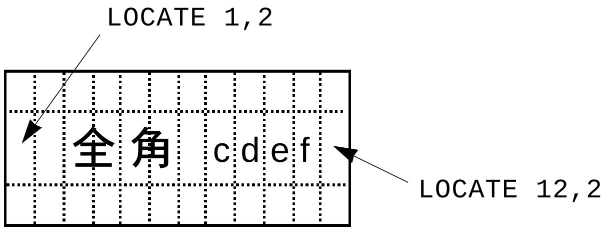
Single-byte ANK Mode (16 columns x 4 lines)



Two-byte Kanji Mode (6 columns x 2 lines for full-width characters only, 12 columns x 2 lines for half-width characters only)

Be careful about the specification of line numbers in the figures below. A single column shown below represents an area for a half-width character; Double columns represent an area for a full-width character.





[2] BHT-4000

Screen mode	Single-byte ANK mode	Two-byte Kanji mode	Condensed two-byte Kanji mode
Columns x Lines	26 x 10	Full-width: 10 x 5 Half-width: 20 x 5	Full-width: 13 x 5 Half-width: 26 x 5

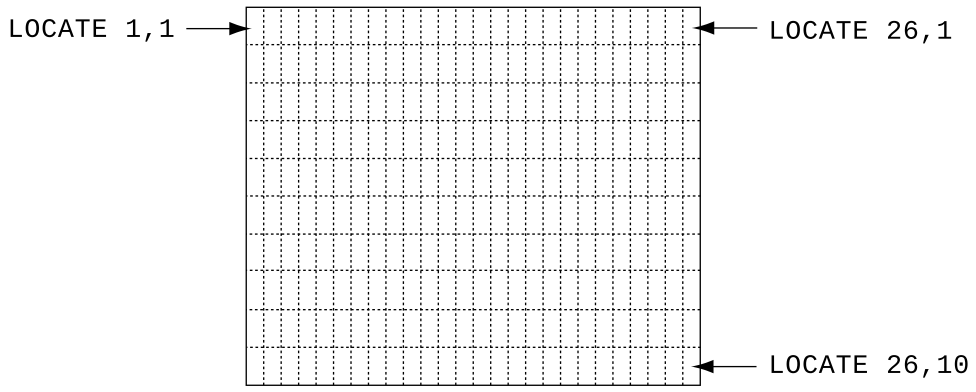
■ Displaying Kanji Characters

To display characters in the condensed two-byte Kanji mode, the BHT-4000 condenses the Kanji patterns of 16 x 16 dots designed for the two-byte Kanji mode. For statements on how to condense, refer to Appendix C3.

■ Locating Characters on the LCD Screen

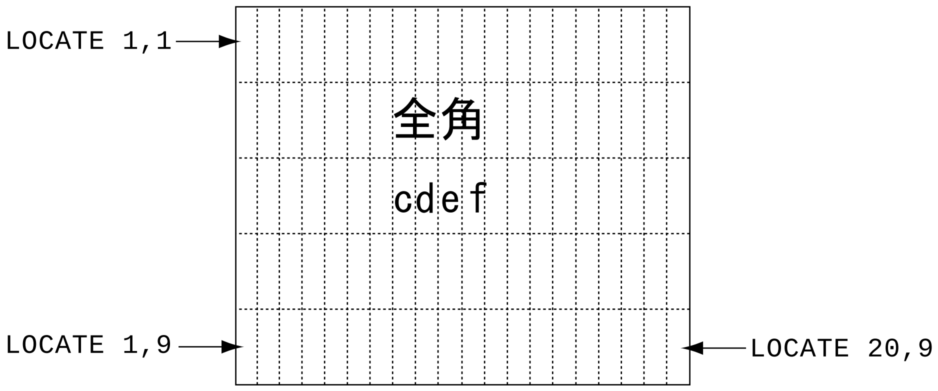
Using the `LOCATE` statement locates characters on the coordinates of the LCD screen. The coordinates differ depending upon the screen mode as shown below.

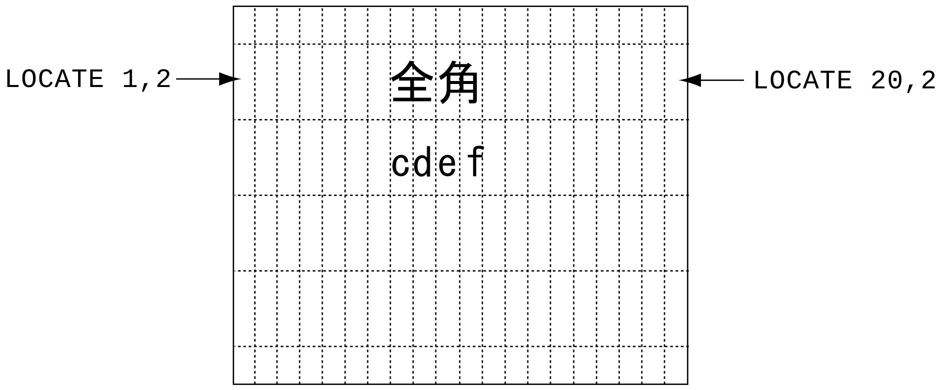
Single-byte ANK Mode (26 columns x 10 lines)



Two-byte Kanji Mode (10 columns x 5 lines for full-width characters only, 20 columns x 5 lines for half-width characters only)

Be careful about the specification of line numbers in the figures below. A single column shown below represents an area for a half-width character; Double columns represent an area for a full-width character.

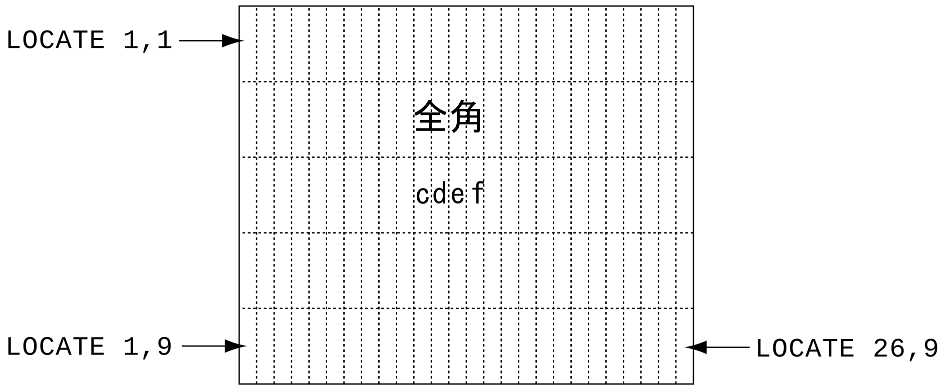


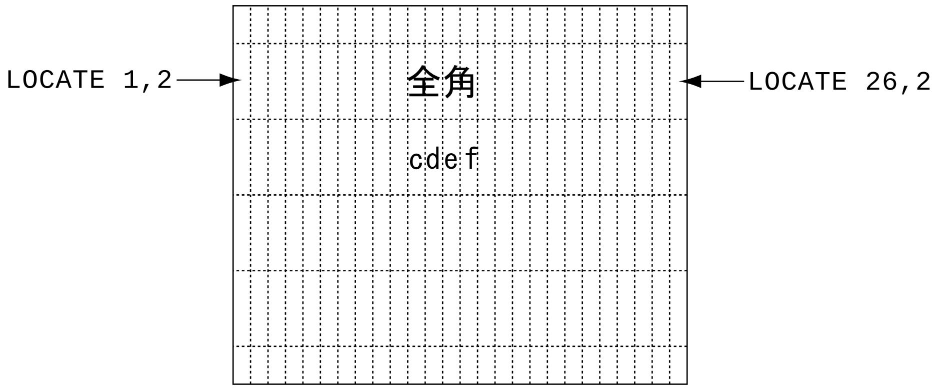


Condensed Two-byte Kanji Mode (13 columns x 5 lines for full-width characters only, 26 columns x 5 lines for half-width characters only)

Be careful about the specification of line numbers in the figures below. A single column shown below represents an area for a half-width character; Double columns represent an area for a full-width character.

(For the Kanji patterns in the condensed two-byte Kanji mode, refer to Appendix C3.)





[3] BHT-5000

Screen mode	Single-byte ANK mode	Two-byte Kanji mode	Condensed two-byte Kanji mode
Columns x Lines	21 x 8	Full-width: 8 x 4 Half-width: 16 x 4	Full-width: 10 x 4 Half-width: 21 x 4

■ Displaying Kanji Characters

To display Kanji characters, it is necessary to download the Kanji font file consisting of JIS Level 1 and Level 2 font files to the BHT-5000 beforehand.

Even without those files, the half-width alphanumerics and Katakana may be displayed.

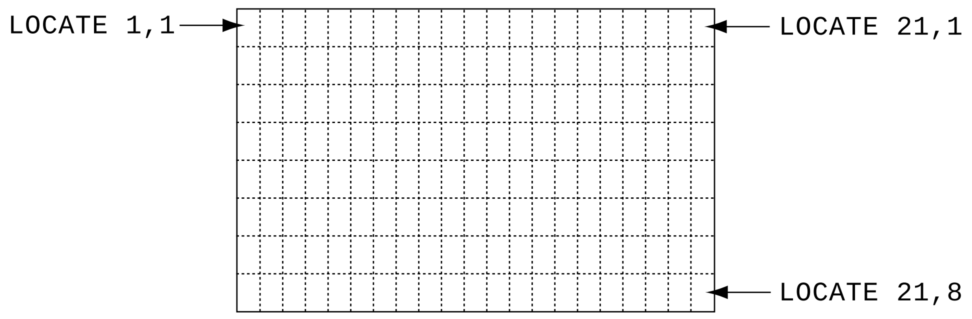
If in user programs you use Kanji characters whose fonts are not downloaded to the BHT-5000, they will appear as "□" on the LCD.

To display characters in the condensed two-byte Kanji mode, the BHT-5000 condenses the Kanji patterns of 16 x 16 dots designed for the two-byte Kanji mode. For statements on how to condense, refer to Appendix C3.

■ Locating Characters on the LCD Screen

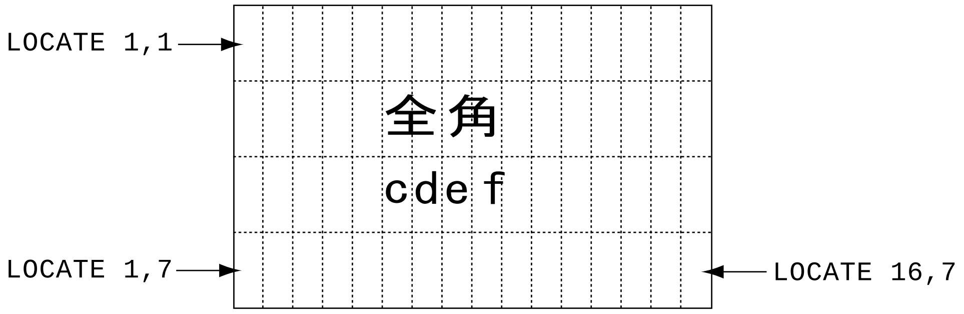
Using the `LOCATE` statement locates characters on the coordinates of the LCD screen. The coordinates differ depending upon the screen mode as shown below.

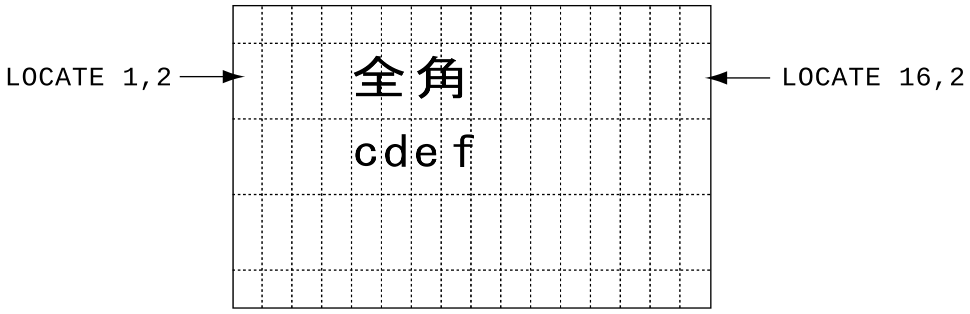
Single-byte ANK Mode (21 columns x 8 lines)



Two-byte Kanji Mode (8 columns x 4 lines for full-width characters only, 16 columns x 4 lines for half-width characters only)

Be careful about the specification of line numbers in the figures below. A single column shown below represents an area for a half-width character; Double columns represent an area for a full-width character.

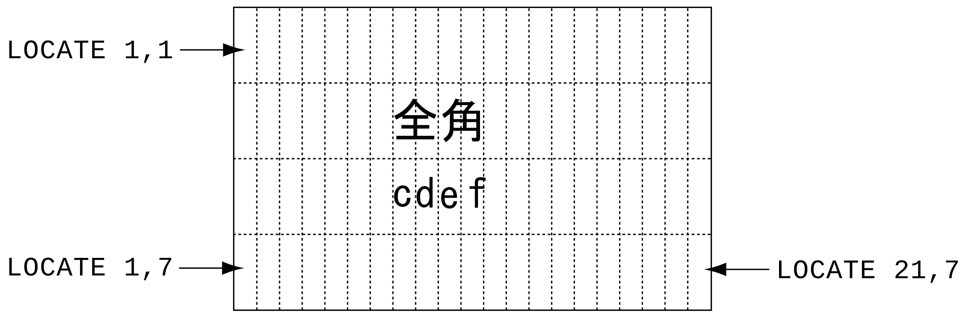


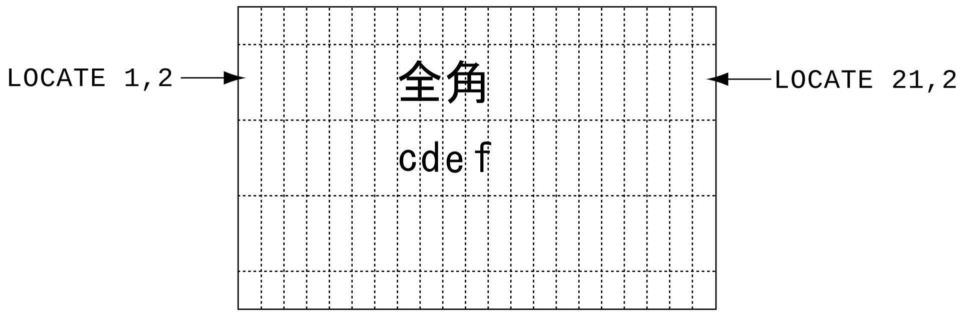


Condensed Two-byte Kanji Mode (10 columns x 4 lines for full-width characters only, 21 columns x 4 lines for half-width characters only)

Be careful about the specification of line numbers in the figures below. A single column shown below represents an area for a half-width character; Double columns represent an area for a full-width character.

(For the Kanji patterns in the condensed two-byte Kanji mode, refer to Appendix C3.)





[4] BHT-6000/BHT-6500

Screen mode	Single-byte ANK mode		Two-byte Kanji mode	
Font size	Standard-size	Small-size	Standard-size	Small-size
Columns x Lines	16 x 10	16 x 8	Full-width: 6 x 3 Half-width: 12 x 3	Full-width: 8 x 4 Half-width: 16 x 4

■ Displaying Kanji Characters

To display Kanji characters, it is necessary to download the Kanji font file consisting of JIS Level 1 and Level 2 font files to the BHT-6000/BHT-6500 beforehand.

Even without those files, the half-width alphanumerics and Katakana may be displayed.

If in user programs you use Kanji characters whose fonts are not downloaded to the BHT-6000/BHT-6500, they will appear as "□" on the LCD.

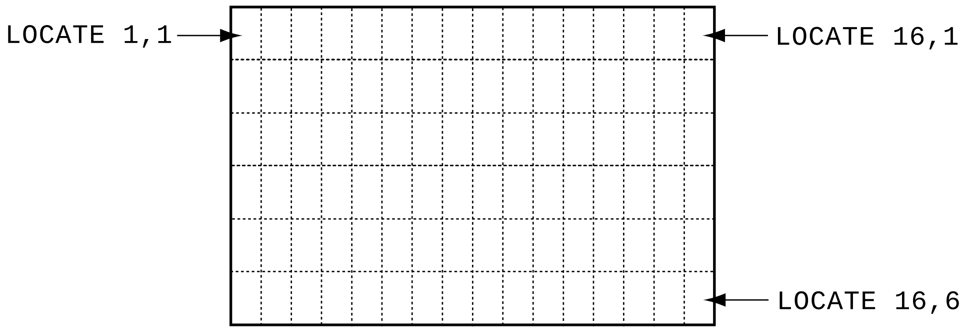
To display characters in the condensed two-byte Kanji mode, the BHT-6000/BHT-6500 condenses the Kanji patterns of 16 x 16 dots designed for the two-byte Kanji mode. For statements on how to condense, refer to Appendix C3.

■ Locating Characters on the LCD Screen

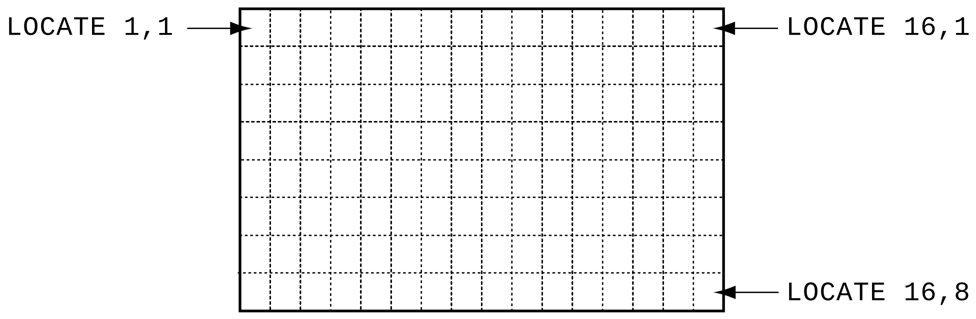
Using the `LOCATE` statement locates characters on the coordinates of the LCD screen. The coordinates differ depending upon the screen mode and the display font size as shown below.

Single-byte ANK Mode

Standard-size font (16 columns x 6 lines)



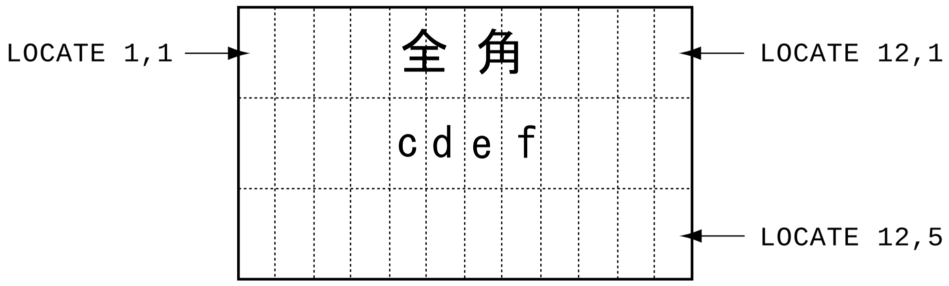
Small-size font (16 columns x 8 lines)

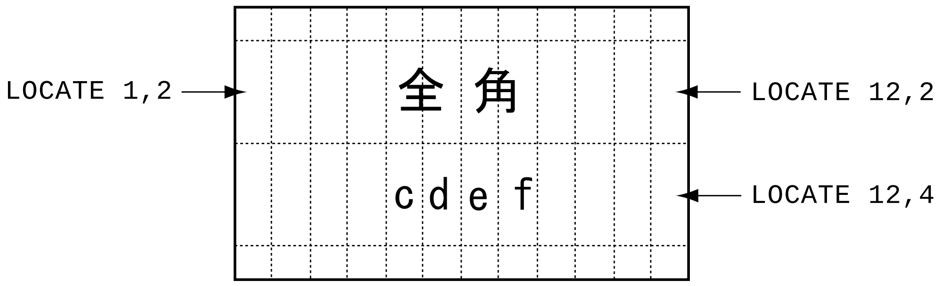


Two-byte Kanji Mode

Standard-size font (6 columns x 3 lines for full-width characters only, 12 columns x 3 lines for half-width characters only)

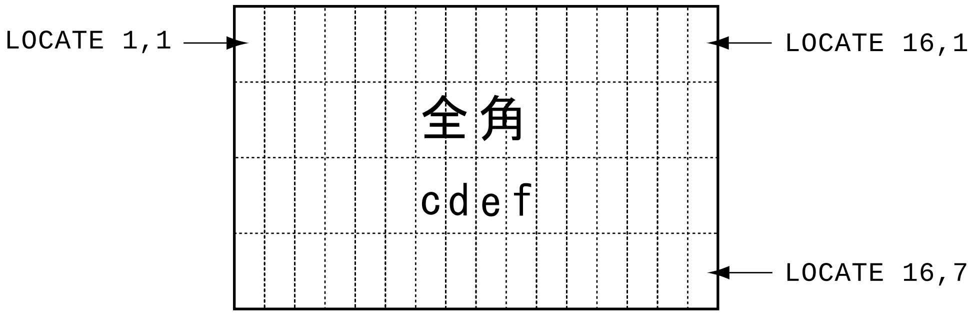
Be careful about the specification of line numbers in the figures below. A single column shown below represents an area for a half-width character: Double columns represent an area for a full-width character.

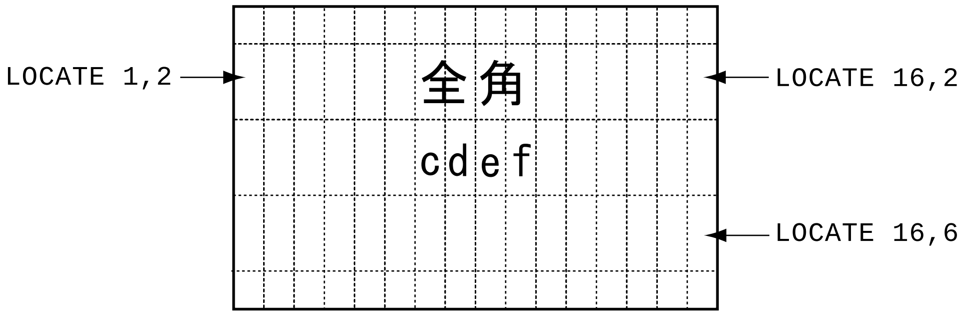




Small-size font (8 columns x 4 lines for full-width characters only, 16 columns x 4 lines for half-width characters only)

Be careful about the specification of line numbers in the figures below. A single column shown below represents an area for a half-width character: Double columns represent an area for a full-width character.





NOTE The small-size fonts of alphanumerics and a part* of the JIS Level 1 Kanji are contained in the JIS Level 1 font file. For other characters whose small-size fonts are not available, the BHT-6000/BHT-6500 condenses the flash-ROMed standard-size font data for display into the small-size of 12 x 12 dots. Some condensed characters might not be legible, so you are recommended to load user-defined fonts (max. 32) for them by using the KPLOAD statement.

*Kanji characters mainly used in the system messages

[5] BHT-7000

(1) Normal-width

Screen mode	Single-byte ANK mode		Two-byte Kanji mode	
Font size	Standard-size	Small-size	Standard-size	Small-size
Columns x Lines	21 x 8	21 x 10	Full-width: 8 x 4 Half-width: 16 x 4	Full-width: 10 x 5 Half-width: 21 x 5

(2) Double-width

Screen mode	Single-byte ANK mode		Two-byte Kanji mode	
Font size	Standard-size	Small-size	Standard-size	Small-size
Columns x Lines	10 x 8	10 x 10	Full-width: 4 x 4 Half-width: 8 x 4	Full-width: 5 x 5 Half-width: 10 x 5

■ Displaying Kanji Characters

To display Kanji characters, it is necessary to download Kanji font files listed below.

- To use standard-size fonts: 16-dot font file
- To use small-size fonts: 12-dot font file

Even without those files, the half-width alphanumerics and Katakana may be displayed.

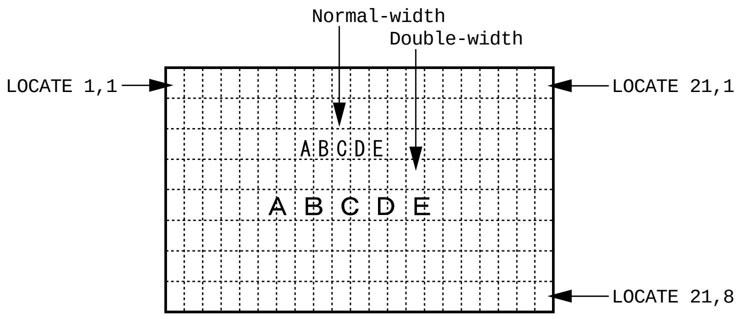
Each of the 16-dot and 12-dot font files consists of JIS Level 1 and Level 2 font files.

■ Locating Characters on the LCD Screen

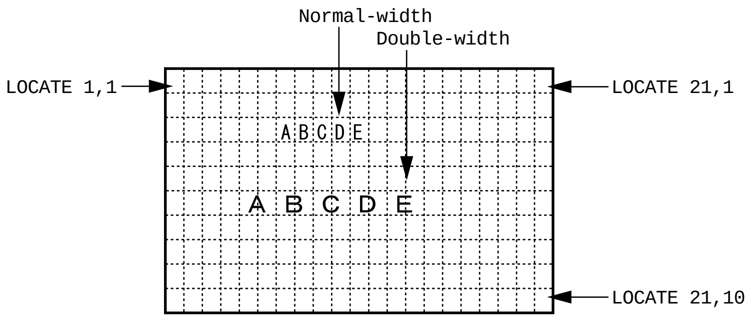
Using the `LOCATE` statement locates characters on the coordinates of the LCD screen. The coordinates differ depending upon the screen mode and the display font size as shown below.

Single-byte ANK Mode

Standard-size font
(21 columns x 8 lines for normal-width,
10 columns x 8 lines for double-width)



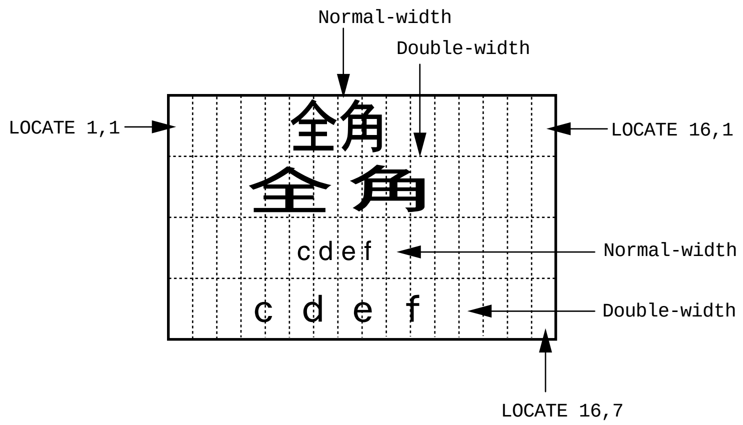
Small-size font
(21 columns x 10 lines for normal-width,
10 columns x 10 lines for double-width)

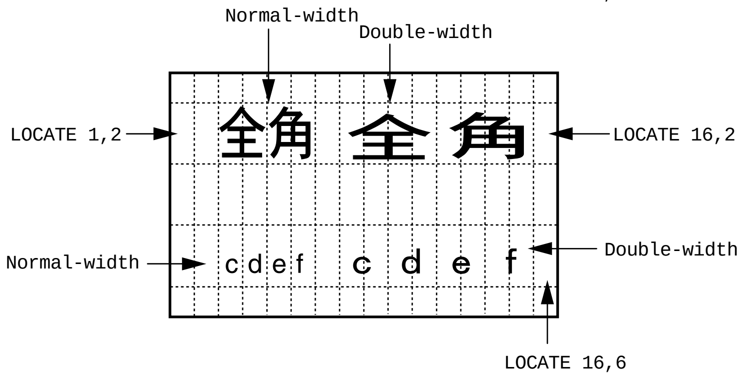


Two-byte Kanji Mode

Standard-size font
8 columns x 4 lines for full-width characters only,
4 columns x 4 lines for full-width characters in double-width mode only,
16 columns x 4 lines for half-width characters only,
8 columns x 4 lines for half-width characters in double-width mode only

Be careful about the specification of line numbers in figures below. A single column shown below represents an area for a half-width character; Double columns represent an area for a half-width character in double-width mode or for a full-width character; Four columns represent an area for a full-width character in double-width mode.

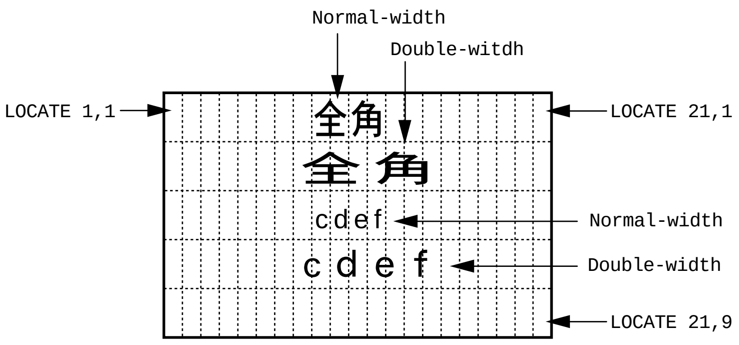


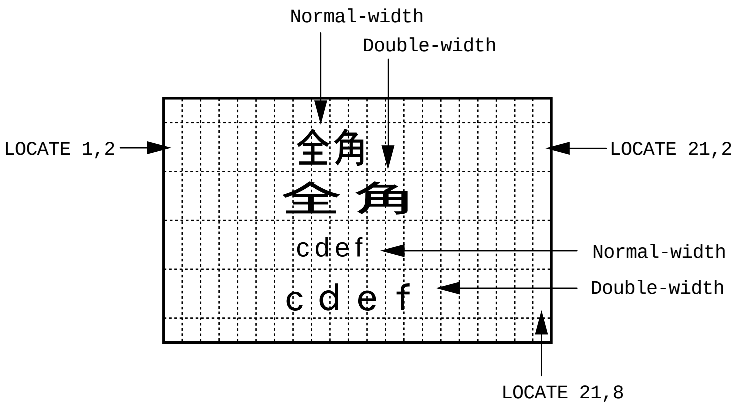


Small-size font

10 columns x 5 lines for full-width characters only,
5 columns x 5 lines for full-width characters in double-width mode only,
21 columns x 5 lines for half-width characters only,
10 columns x 5 lines for half-width characters in double-width mode only

Be careful about the specification of line numbers in figures below. A single column shown below represents an area for a half-width character; Double columns represent an area for a half-width character in double-width mode or for a full-width character; Four columns represent an area for a full-width character in double-width mode.





[6] BHT-7500

(1) Normal-width

Screen mode	Single-byte ANK mode		Two-byte Kanji mode	
Font size	Standard-size	Small-size	Standard-size	Small-size
Columns x Lines	26 x 20	26 x 26	Full-width: 10 x 10 Half-width: 20 x 10	Full-width: 13 x 13 Half-width: 26 x 13

(2) Double-width

Screen mode	Single-byte ANK mode		Two-byte Kanji mode	
Font size	Standard-size	Small-size	Standard-size	Small-size
Columns x Lines	13 x 20	13 x 26	Full-width: 5 x 10 Half-width: 10 x 10	Full-width: 6 x 13 Half-width: 13 x 13

■ Displaying Kanji Characters

To display Kanji characters, it is necessary to download Kanji font files listed below.

- To use standard-size fonts: 16-dot font file
- To use small-size fonts: 12-dot font file

Even without those files, the half-width alphanumerics and Katakana may be displayed.

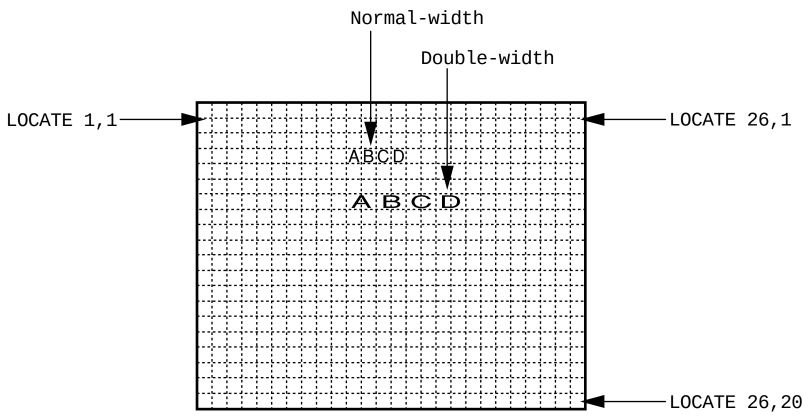
Each of the 16-dot and 12-dot font files consists of JIS Level 1 and Level 2 font files.

■ Locating Characters on the LCD Screen

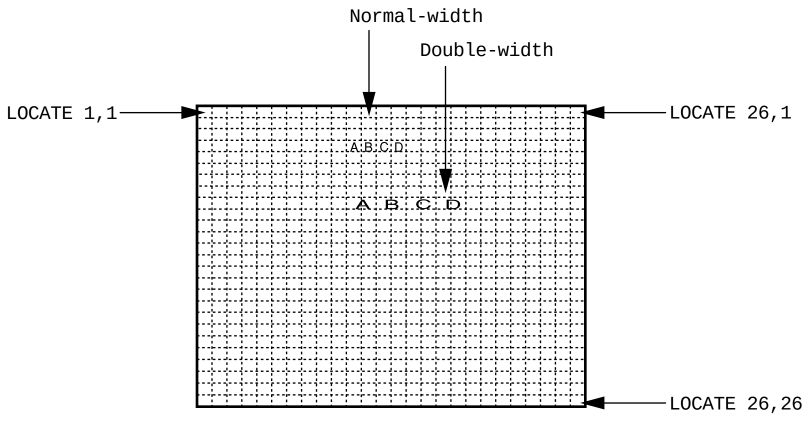
Using the LOCATE statement locates characters on the coordinates of the LCD screen. The coordinates differ depending upon the screen mode and the display font size as shown below.

Single-byte ANK Mode

Standard-size font
(26 columns x 20 lines for normal-width,
13 columns x 20 lines for double-width)



Small-size font
(26 columns x 26 lines for normal-width,
13 columns x 26 lines for double-width)

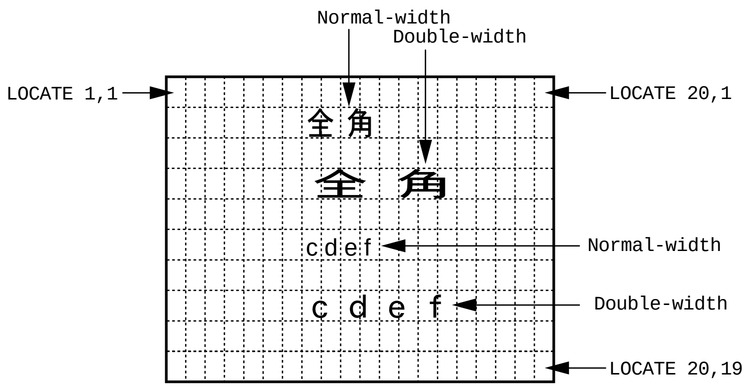


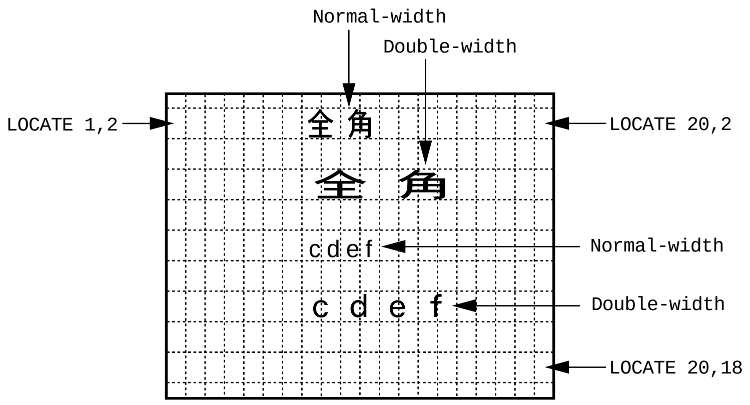
Two-byte Kanji Mode

<u>Standard-size font</u>

10 columns x 10 lines for full-width characters only,
5 columns x 10 lines for full-width characters in double-width mode only,
20 columns x 10 lines for half-width characters only,
10 columns x 10 lines for half-width characters in double-width mode only

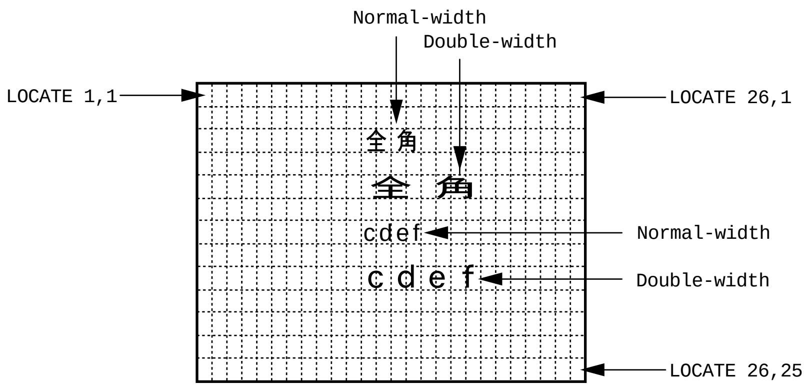
Be careful about the specification of line numbers in figures below. A single column shown below represents an area for a half-width character; Double columns represent an area for a half-width character in double-width mode or for a full-width character; Four columns represent an area for a full-width character in double-width mode.

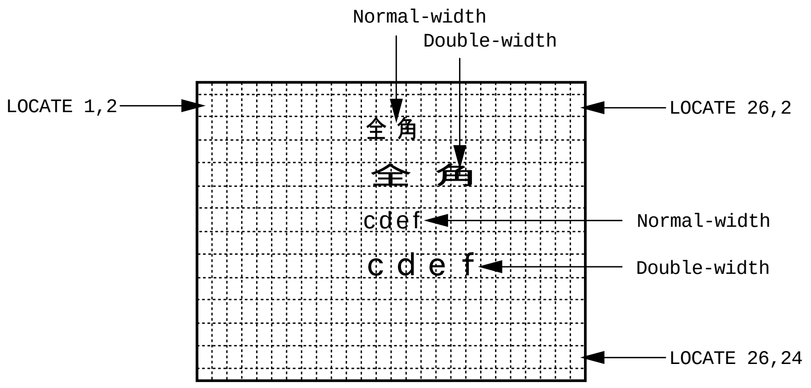




Small-size font	13 columns x 13 lines for full-width characters only, 6 columns x 13 lines for full-width characters in double-width mode only, 26 columns x 13 lines for half-width characters only, 13 columns x 13 lines for half-width characters in double-width mode only

Be careful about the specification of line numbers in figures below. A single column shown below represents an area for a half-width character; Double columns represent an area for a half-width character in double-width mode or for a full-width character; Four columns represent an area for a full-width character in double-width mode.





7.1.3 Dot Patterns of Fonts

■ Character fonts

In the figures below, "■" shows a display area for characters. Any character is displayed within a set of the display areas.

"□" shows a delimiter area that separates characters from each other and contains no display data. The corresponding dots are always off.

The double-width mode is supported by the BHT-7000/BHT-7500.

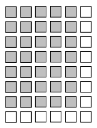
Small-size fonts are supported by the BHT-6000/BHT-6500/BHT-7000/BHT-7500.

The condensed two-byte Kanji mode is supported by the BHT-4000/BHT-5000.

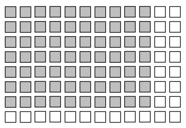
Single-byte ANK Mode

Standard-size font

6 x 8 dots

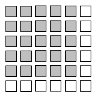


12 x 8 dots (in double-width mode)

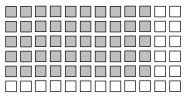


Small-size font

6 x 6 dots



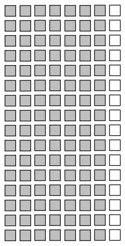
12 x 6 dots (in double-width mode)



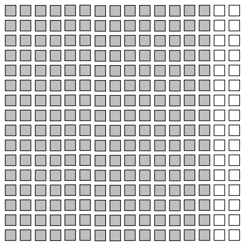
Two-byte Kanji Mode[*1]

Standard-size font

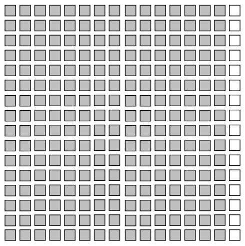
Half-width Kanji
8 x 16 dots



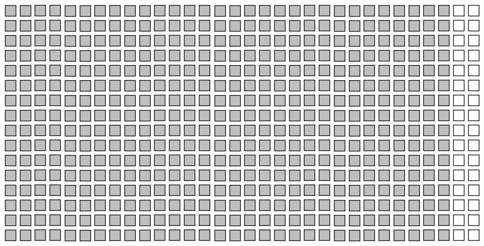
Half-width Kanji (in double-width mode)
16 x 16 dots



Full-width Kanji
16 x 16 dots

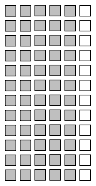


Full-width Kanji (in double-width mode)
32 x 16 dots

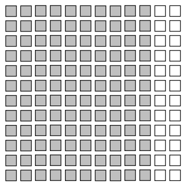


Small-size font

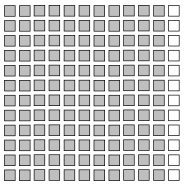
Half-width Kanji
6 x 12 dots



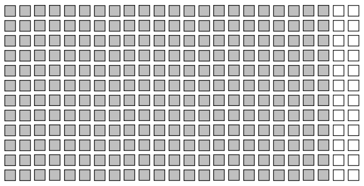
Half-width Kanji (in double-width mode)
12 x 12 dots



Full-width Kanji
12 x 12 dots

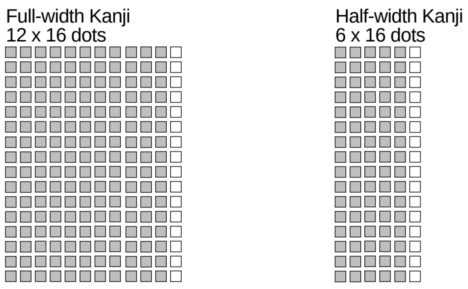


Full-width Kanji (in double-width mode)
24 x 12 dots



[*1] The BHT-7000/BHT-7500 fonts do not use the lowermost dot line of the letter frame.

Condensed two-byte Kanji Mode



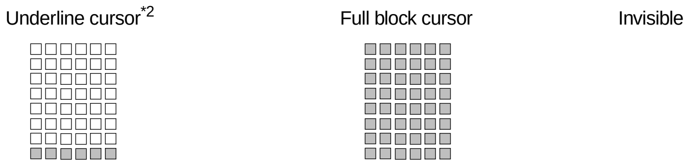
■ Cursor shape

The LOCATE statement specifies the cursor shape--Underline cursor, full block cursor, or invisible.

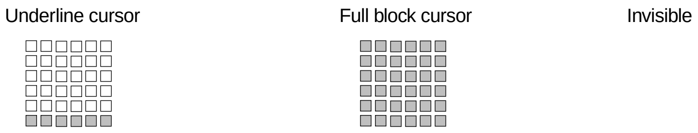
In the BHT-7000/BHT-7500, you may define and load the desired cursor shape with the APLOAD or KPLOAD statement and then specify the user-defined cursor with the LOCATE statement. In the double-width screen mode, the cursor will be displayed in double width.

Single-byte ANK Mode

Standard-size font (6 x 8 dots)



Small-size font (6 x 6 dots)

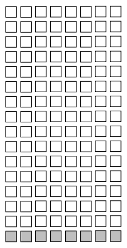


*2 In the BHT-7500, the underline cursor of the standard-size font will be displayed not on the lowermost dot line of the letter frame but on the 2nd dot line from the bottom. This is because the BHT-7500 uses the lowermost dot line for showing the system status.

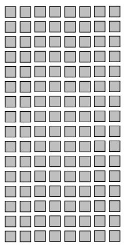
Two-byte Kanji Mode

Standard-size font (8 x 16 dots)

Underline cursor*2



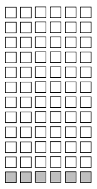
Full block cursor



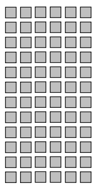
Invisible

Small-size font (6 x 12 dots)

Underline cursor



Full block cursor



Invisible

*2 In the BHT-7500, the underline cursor of the standard-size font will be displayed not on the lowermost dot line of the letter frame but on the 2nd dot line from the bottom. This is because the BHT-7500 uses the lowermost dot line for showing the system status.

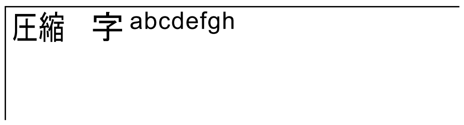
7.1.4 Mixed Display of Different Character Types or Different-size Fonts

[1] Displaying ANK, Kanji, and Condensed Kanji in One Line

It is possible to mix-display the ANK characters, Kanji characters (both full-width and half-width), and condensed Kanji characters (both full-width and half-width) in the same line on the LCD screen, as shown in the example below.

```
CLS
SCREEN 0
LOCATE 1,1 : PRINT "ABCDEFGHabcdefgh"
SCREEN 1
LOCATE 1,1 : PRINT "漢　字"
SCREEN 2
LOCATE 1,1 : PRINT "圧縮"
```

If the display data is outputted to the same location more than one time as shown in the above program, the BHT overwrites the old data with new data.

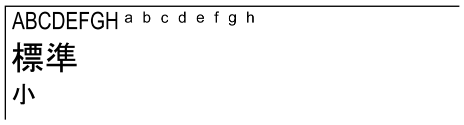


```
圧縮　字abcdefgh
```

[2] Displaying Standard- and Small-size Fonts on the Same Screen

The BHT-6000/BHT-6500 can mix-display the standard- and small-size fonts of ANK characters and Kanji characters (both full width and half-width) on the same screen.

```
CLS
OUT &h6080, 0                  'Selects standard-size font
SCREEN 0
PRINT "ABCDEFGH";
OUT &h6080, 1                  'Selects small-size font
PRINT "abcdefgh"
OUT &h6080, 0                  'Selects standard-size font
SCREEN 1
LOCATE 1,2  :PRINT "標準"
OUT &h6080, 1                  'Selects small-size font
PRINT "小";
```

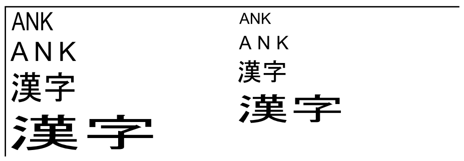


```
ABCDEFGHa b c d e f g h
標準
小
```

[3] Displaying Normal- and Double-width Characters on the Same Screen

The BHT-7000/BHT-7500 can mix-display the normal- and double-width characters on the same screen.

```
CLS
OUT &h6080,0                                 ’ Standard-size font
SCREEN 0,0 : PRINT "ANK"                     ’ Normal-width in single-byte ANK mode
SCREEN 0,2 : PRINT "ANK"                     ’ Double-width in single-byte ANK mode
SCREEN 1,0 : PRINT " 漢字 "                   ’ Normal-width in two-byte Kanji mode
SCREEN 1,2 : PRINT " 漢字 "                   ’ Double-width in two-byte Kanji mode
LOCATE 1,1
OUT &h6080,1                                 ’ Small-size font
SCREEN 0,0 : LOCATE 14 : PRINT "ANK"         ’ Normal-width in single-byte ANK mode
SCREEN 0,2 : LOCATE 14 : PRINT "ANK"         ’ Double-width in single-byte ANK mode
SCREEN 1,0 : LOCATE 14 : PRINT " 漢字 "’      Normal-width in two-byte Kanji mode
SCREEN 1,2 : LOCATE 14 : PRINT " 漢字 "’      Double-width in two-byte Kanji mode
```



```
ANK          ANK
A N K        A N K
漢字          漢字
漢 字         漢 字
```

7.1.5 Displaying User-defined Characters

■ Loading a user-defined font

The APLOAD or KPLOAD statement loads a user-defined font.

The APLOAD statement is capable of loading up to 32 single-byte ANK fonts to be displayed in the single-byte ANK mode.

The KPLOAD statement is capable of loading up to 32 or 128 two-byte Kanji fonts (depending on the BHT series as listed below) in full width to be displayed in the two-byte Kanji mode or condensed two-byte Kanji mode.

- 32: BHT-3000/BHT-4000/BHT-5000/BHT-6000/BHT-6500
- 128: BHT-7000/BHT-7500

■ Condensing/enlarging the defined font for display

If the condensed two-byte Kanji mode, small-size font, or double-width is specified, then the BHT condenses or enlarges user-defined fonts loaded by the APLOAD or KPLOAD to display. (For details about condensing, refer to Appendix C3.)

7.1.6 VRAM

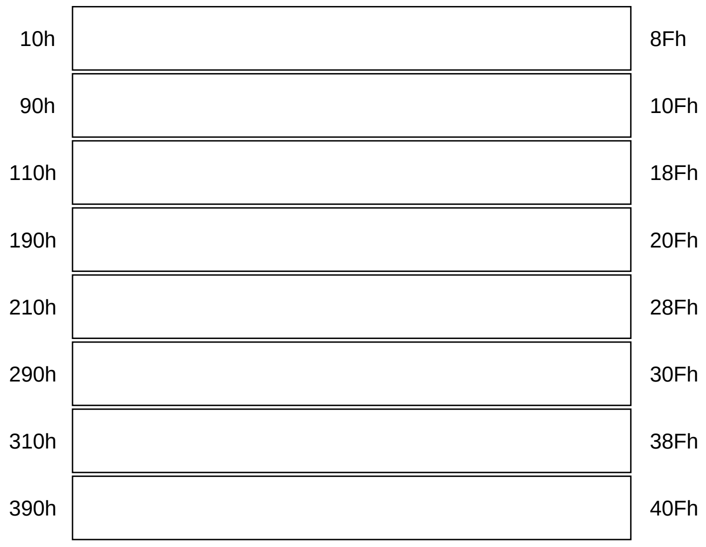
The `INP` function may read the VRAM data. The `OUT` statement writes data into the VRAM so that graphics may be displayed on the LCD dotwise.

■ Specifying an address bytewise

An address on the LCD may be specified bytewise by giving a port number in the `OUT` statement and `INP` function. The entry range of the port number is as follows:

Series	Entry range of the port number
BHT-3000	10h to 18Fh
BHT-4000	10h to 64Fh
BHT-5000	10h to 40Fh
BHT-6000	10h to 24Fh
BHT-6500	10h to 24Fh
BHT-7000	10h to 40Fh
BHT-7500	10h to C7Fh

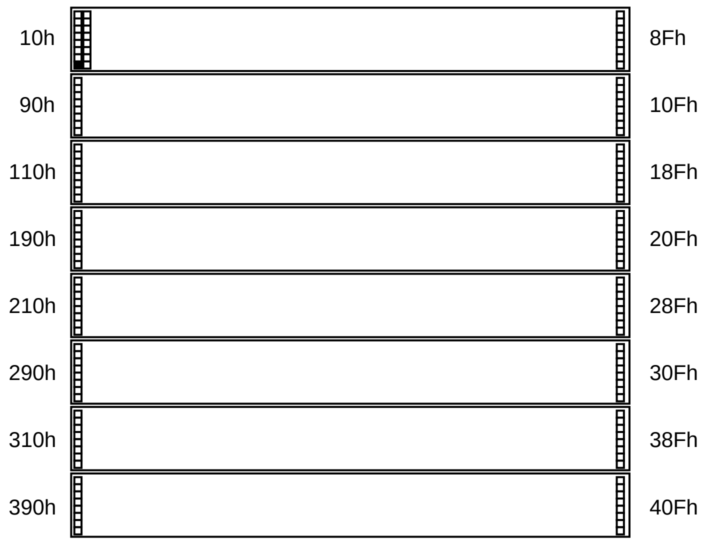
Port numbering system counts, starting from the top left corner of the LCD to the right bottom corner. The figure below shows the port numbers available on the BHT-7000.



■ Setting an 8-bit binary pattern

The data of an 8-bit binary pattern should be designated by bit 7 (LSB) to bit 0 (MSB). If the bit is 1, the corresponding dot on the LCD will come ON.

```
OUT &h10,&h80      'Set bit 7 only to 1
```



In the BHT-7500, you may set graphic data to the VRAM area assigned to the bottom dot line of the LCD by using the `OUT` statement. The set data cannot be displayed on the LCD but can be read out with the `INP` function. Scrolling the screen will also display the data set on the bottom dot line.

7.1.7 Displaying the System Status (BHT-4000/BHT-5000/BHT-6000/BHT-6500)

The BHT-4000 may display the voltage level icon and shifted key icon on the bottom line of the LCD.

The BHT-5000/BHT-6000/BHT-6500 may display the shifted key icon and alphabet input icon at the right end of the bottom line of the LCD.

For details about the icon shapes, refer to the BHT's User's Manual.

[1] BHT-4000

■ Turning the system status indication on or off

You may turn the system status indication on or off on the SET DISPLAY menu in System Mode. The default is ON. (For the setting procedure, refer to the "BHT-4000 User's Manual.") You may control the system status indication also by using the `OUT` statement in user programs. (Refer to Appendix D, "II/O Ports.")

■ Number of lines controllable when the system status is displayed

Setting the system status indication to ON occupies the bottom line of the LCD, so the number of lines controllable by user programs decreases by one line as listed below. (The number of columns undergoes no change.)

Screen Mode	Character Type	Number of Lines Controllable by User Programs
Single-byte ANK mode	ANK characters	9 lines
Two-byte Kanji mode	Full-width Kanji	4 lines
	Half-width Kanji	4 lines
Condensed two-byte Kanji mode	Full-width Kanji	4 lines
	Half-width Kanji	4 lines

■ Notes relating to the system status

Notes when the system status is displayed

The following statements and functions will cause somewhat different operations when the system status is displayed.

• `CLS` statement

The `CLS` statement clears the VRAM area assigned to the bottom line of the LCD but does not erase the system status displayed.

• `LOCATE` statement

Even if you specify the bottom line of the LCD as the desired cursor position by using the `LOCATE` statement, the cursor cannot move to the bottom line and it will move to the next to the bottom line instead.

• `OUT` statement

If you send graphic data to the VRAM area assigned to the bottom line of the LCD by using the `OUT` statement, the sent data will be written into that VRAM area but cannot be displayed on the bottom line.

• `INP` function

If you specify the VRAM area assigned to the bottom line of the LCD as an input port, the `INP` function reads one-byte data from that area.

Notes when displaying the system status with OUT statement

If the cursor is placed on any line except for the bottom line of the LCD: Specifying the system status indication with the `OUT` statement overwrites the system status on the current data shown on the bottom line. If Kanji characters are shown on the bottom line, the lower half of the Kanji is overwritten with the system status but with the upper half remaining on the LCD.

If the cursor is placed on the bottom line of the LCD: Specifying the system status indication with the `OUT` statement scrolls up the screen by one line together with the cursor and the system status will appear on the new bottom line. (The number of columns does not change.)

Notes when erasing the system status with the OUT statement

Erasing the system status with the `OUT` statement displays the content of the VRAM area (assigned to the bottom line of the LCD) on that part of the LCD.

[2] BHT-5000/BHT-6000/BHT-6500

■ Turning the system status indication on or off

You may turn the system status indication on or off on the SET DISPLAY menu in System Mode. The default is ON. (For the setting procedure, refer to the "BHT's User's Manual") You may control the system status indication also by using the `OUT` statement in user programs. (Refer to Appendix D, "I/O Ports.")

■ Notes relating to the system status

Notes when the system status is displayed

The following statements and functions will cause somewhat different operations when the system status is displayed.

- `CLS` statement

The `CLS` statement clears the VRAM area assigned to the right end of the bottom line of the LCD but does not erase the system status displayed.

- `OUT` statement

If you send graphic data to the VRAM area assigned to the right end of the bottom line of the LCD by using the `OUT` statement, the sent data will be written into that VRAM area but cannot be displayed on the bottom line.

- `INP` function

If you specify the VRAM area assigned to the right end of the bottom line of the LCD as an input port, the `INP` function reads one-byte data from that area.

Notes when displaying the system status with OUT statement

Specifying the system status indication with the `OUT` statement overwrites the system status on the current data shown at the right end of the bottom line of the LCD. If Kanji characters are shown at the right end of the bottom line, the lower half of the Kanji is overwritten with the system status but with the upper half remaining on the LCD.

Notes when erasing the system status with the OUT statement

Erasing the system status with the `OUT` statement displays the content of the VRAM area (assigned to the right end of the bottom line of the LCD) on that part of the LCD.

7.1.8 Other Facilities for the LCD

■ Setting national characters

Using the `COUNTRY$` function displays currency symbols and special characters for countries in the screen mode below.

• Single-byte ANK mode:	All BHT series
• Two-byte Kanji mode (half-width):	BHT-7000/BHT-7500

Refer to Appendix C2, "National Character Sets."

■ Highlighting characters

The `SCREEN` statement highlights characters.

Display		`SCREEN` statement
Regular display	Normal-width	`SCREEN ,0`
	Double-width	`SCREEN ,2` (See Note below.)
Highlighted display	Normal-width	`SCREEN ,1`
	Double-width	`SCREEN ,3` (See Note below.)

Note: Supported by the BHT-7000/BHT-7500 only.

■ Specifying the cursor shape

The `LOCATE` statement specifies the cursor shape.

Cursor shape	`LOCATE` statement
Invisible	`LOCATE ,,0`
Underline cursor	`LOCATE ,,1`
Full block cursor	`LOCATE ,,2`
User-defined cursor	`LOCATE ,,255` (See Note below.)

Note: Supported by the BHT-7000/BHT-7500 only.

The shape of a user-defined cursor may be defined by using the `APLOAD` or `KPLOAD` statement in the single-byte ANK mode or two-byte Kanji mode, respectively.

In the single-byte ANK mode, the cursor size will become equal to the size of single-byte ANK characters; in the double-byte Kanji mode or condensed double-byte Kanji mode, it will become equal to the size of the half-width characters in each mode.

7.2 Input from the Keyboard

7.2.1 Function Keys

Any of the following operations makes the pressed key act as a function key:

- Pressing one of the function keys. *1
- Pressing one of the function keys while holding down the Shift key. *2
- Pressing one of the numeric keys while holding down the Shift key. *2

*1 Since each of the function keys is assigned its default value of a character code or control code, pressing it enters the default value. New assignment is possible with a KEY statement as described below.

*2 If pressed with the Shift key held down, not only the function keys but also numeric keys serve as function keys.

For the keyboard layouts, key numbers, and key assignments, refer to Appendix E, "Key Number Assignment on the Keyboard."

■ Assigning a character string to a function key

You can assign a desired character string (up to two characters) or a single control code to a function key by using the KEY statement, as shown below.

- Example for characters

```
KEY 1,"AB"
```

- Example for a control code

```
KEY 2,CHR$(8)     '-Backspace
```

Where a backspace code is assigned to the function key numbered 2.

NULL Character or String Assignment

Assigning a NULL character or string to a function key makes the entry of that function key invalid if pressed. In the example below, pressing the keys numbered 3 and 4 produces no keyboard entry.

```
KEY 3,""
KEY 4,CHR$(0)
```

■ Defining a function key as the LCD backlight function on/off key

You can define a particular function key as the backlight function on/off key and set the length of backlight ON-time by using the KEY statement, as shown below.

- Example for defining the key numbered 5 and setting 60 seconds.

```
KEY 5,"BL60"
```

NOTE It is impossible to assign both a character string and the backlight on/off function to a same function key. For details, refer to KEY in Chapter 14.

■ Defining a magic key

<u>**BHT-5000/BHT-6000/BHT-6500**</u>

You can define a magic key as the SF key, trigger switch, or battery voltage display key, as well as assigning a character string, control code, ENT key, or backlight function on/off key to it. (In the BHT-6000, the trigger switch function is assigned to both M1 and M2 keys by default; in the BHT-6500, it is assigned to all of M1 to M4 keys by default.)

- Example for defining the M1 key as the SF key.

```
KEY 30,"SFT"
```

- Example for defining the M2 key as the trigger switch.

```
KEY 31,"TRG"
```

- Example for defining the M1 key as the battery voltage display key.

```
KEY 30,"BAT"
```

<u>**BHT-7000/BHT-7500**</u>

You can define a magic key as the SF key or trigger switch, as well as assigning a character string, control code, ENT key, or backlight function on/off key to it. (The trigger switch function is assigned to both M3 and M4 keys by default.)

7.2.2 Keystroke Trapping

You can trap the pressing of a particular key, by programming with the KEY ON, KEY OFF, and ON KEY...GOSUB statements.

NOTE If you specify a function key which has been defined as the LCD backlight function on/off key, trigger switch, shift key, or battery voltage display key for keystroke trapping, no keystroke trap takes place.

For details about the keystroke trapping, refer to Chapter 9, "Section 9.2, "Event Polling."

7.2.3 Alphabet Entry Function

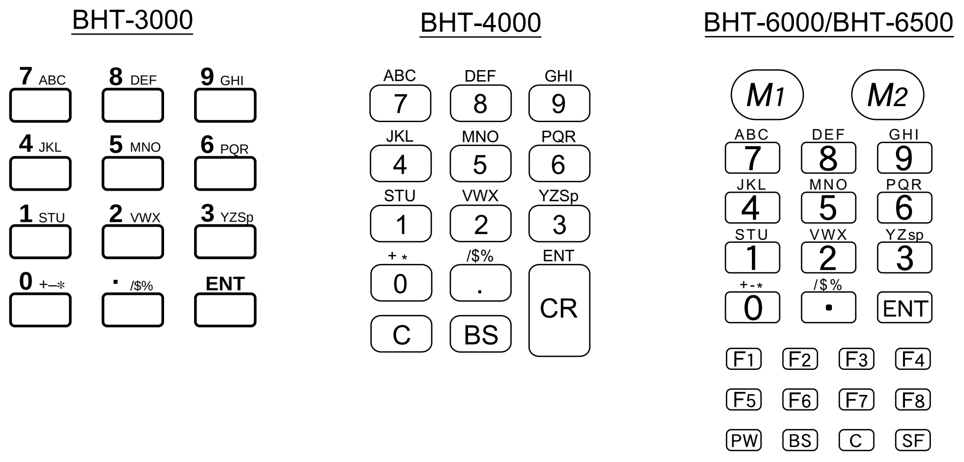
The alphabet entry function allows you to enter alphabetic characters, a space, and symbols from the BHT keyboard (keypad) during execution of a user program.

[1] BHT-3000/BHT-4000/BHT-6000/BHT-6500

To activate or deactivate the alphabet entry function, use `OUT` statement in a user program.

As shown below, three characters are assigned to each of 0-9 numerical keys and period key. For example, A, B, and C are assigned to the 7 key. To designate one of the three assigned characters, use the trigger switch.*

* In the BHT-6000/BHT-6500, use the M1 or M2 key when the trigger switch function is assigned to the key.



■ Alphabet Entry Procedure

(1) Activating the alphabet entry function with `OUT` statement

Issue the `OUT` statement as shown below in a user program.

```
OUT 5, 1
```

NOTE By issuing the `OUT` statement which sets 1 or 0 to bit 0 of port 5, you can activate or deactivate the alphabet entry function, respectively.

To enable:	`OUT 5, &h1`
To disable:	`OUT 5, &h0`

The default setting of the alphabet entry function is "deactivated."

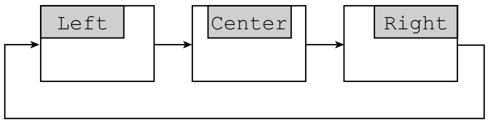
(2) Entering alphabetic characters from the keypad

1) Find a target key which is assigned an alphabetic character to be inputted, and then check the position of the character (Left, Center, or Right) relative to the three characters assigned to the target key.

2) Designate the character position by using the trigger switch and then press the target key.

How to use the trigger switch

Pressing the trigger switch cycles through the shift guidance block $\boxed{\text{Left}}$, $\boxed{\text{Center}}$, and $\boxed{\text{Right}}$ on the LCD as shown below.



The shift guidance block will appear on the top or bottom line, depending upon the current cursor position. That is, if the cursor lies on any of the lower lines, the shift guidance block will appear on the top line; if it lies on any of the upper lines, the block will appear on the bottom line.

The shift guidance block appears only while the trigger switch is held down. Therefore, you should press the target key while holding down the trigger switch.

To enter an N character, for example, use the trigger switch to display the block $\boxed{\text{Center}}$ on the LCD. While displaying the $\boxed{\text{Center}}$, press the 5 key.

During the above entry operation, you can use the Clear, Backspace, and numerical keys as usual.

■ Notes

- In the BHT-3000/BHT-4000, the alphabet entry function is available only in the single-byte code (ANK) mode.
- For displaying the shift guidance block $\boxed{\text{Right}}$ when the status indication is set to ON, the BHT-4000/BHT-6000/BHT-6500 overwrites the status indication with the shift guidance block.
- The activated or deactivated state of the alphabet entry function will be resumed. The shift guidance block will not be resumed.
- User programs cannot distinguish between a character entered with the alphabet entry function and a character generated by pressing a function key which has been assigned the character by the KEY statement, if those characters are the same. (Refer to Subsection 7.2.1, "Function Keys.")

In the example below, the character "A" may be entered with the alphabet entry function or may be generated by pressing the F1 key which has been assigned that character by the KEY statement. The user program, however, cannot distinguish between them so as to transfer control to the program step labelled FUNC1 in both cases.

```
K$=INPUT$ (1)
IF K$="A" THEN GOTO FUNC1 ENDIF
⋮
```

To prevent such a problem, assign any other character to the F1 key with the KEY statement and then modify the judgement condition. For example, replace the character assigned to the F1 key with the character "#", as shown below.

```
KEY 1, "#"
⋮
K$=INPUT$ (1)
IF K$="#" THEN GOTO FUNC1 ENDIF
⋮
```

For details, refer to Chapter 14, KEY and ON KEY statements.

Note that the alphabet entry function does not influence the keystroke trapping which identifies keys according to their key numbers.

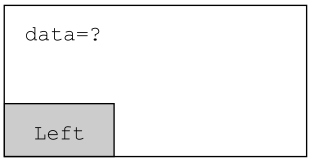
■ Alphabet Entry Example

Coding in a user program:

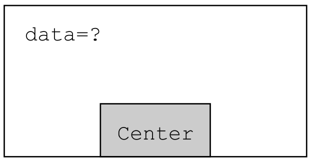
```
OUT 5,1              ’Activating the alphabet
                     ’entry function
INPUT "data=";a$     ’Waiting for keystrokes
```

Entering alphabet characters "ND" under the above user program:

1) Press the trigger switch.

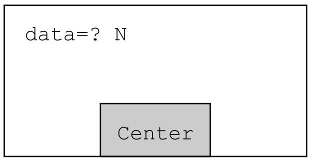


2) Hold down the trigger switch.



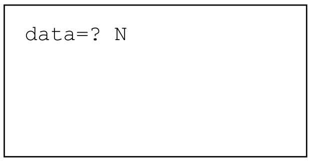
```
data=?
```

3) Without releasing the trigger switch, press the 5 key.



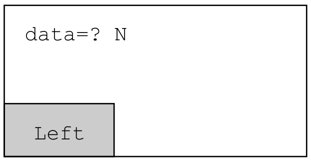
```
data=? N
```

4) Release the trigger switch.



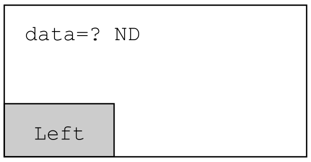
```
data=? N
```

5) Hold down the trigger switch.



```
data=? N
```

6) Without releasing the trigger switch, press the 8 key.



```
data=? ND
```

7) Release the trigger switch.

```
data=? ND
```

8) Press the Enter key to complete the entry operation.

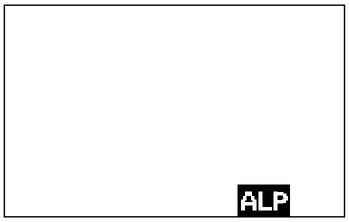
[2] BHT-5000/BHT-7000/BHT-7500 (32-key pad models)

The BHT-5000/BHT-7000/BHT-7500 with a 32-key pad supports the alphabet entry function which can be activated by pressing the ALP key. To deactivate it, press the ALP key again.

To enter lowercase letters in the alphabet input mode, shift the keypad with the SF key. Letter assignment to the keys is shown in Appendix E.

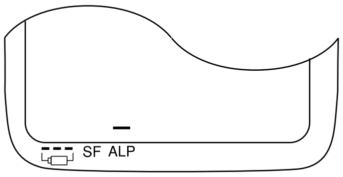
■ BHT-5000

When the alphabet input function is activated, the icon ALP appears at the right end of the bottom line of the LCD as shown at right if you have turned on the system status indication in System Mode.



■ BHT-7000/7500

When the alphabet entry system is selected, a bar appears above the ALP as shown at right.



[3] BHT-7000 (26-key pad model)

In addition to the numeric entry from the keypad, the BHT-7000 with a 26-key pad supports alphabet entry.

■ Switching between the Numeric Entry System and Alphanumeric Entry System

To switch between the numeric entry system and alphanumeric entry system, use the `OUT` statement in a user program as shown below.

```
OUT &h60B0,0       'Switches to the numeric entry system*
OUT &h60B0,1       'Switches to the alphanumeric entry system
```

*Selected when the BHT-7000 is cold-started.

To monitor the current key entry system, use the `INP` function as shown below.

```
INP(&h60B0)
```

■ Switching between Numeric and Alphabet Entry Modes in the Alphanumeric Entry System

In the alphanumeric entry system, you may switch between numeric and alphabet entry modes as described below. The default, which is applied immediately after the BHT-7000 is switched to the alphanumeric entry system, is the numeric entry mode.

• Pressing the SF key

Pressing the SF key toggles between the numeric and alphabet entry modes.

• Using the `OUT` statement

Issue the `OUT` statement as shown below.

```
OUT &h60B1,0       'Switches to the numeric entry mode
OUT &h60B1,1       'Switches to the alphabet entry mode
```

To monitor the current entry mode, use the `INP` function as shown below.

```
INP(&h60B1)
```

■ Alphabet Entry Procedure

(1) Switch to the alphanumeric entry system as follows:

Issue "`OUT &h60B0,1`".

(2) Switch to the alphabet entry mode as follows:

Press the SF key or issue "`OUT &h60B1,1`".

The ALP status bar appears.

(3) Enter alphabet letters from the keypad as follows:

1) Press a numerical key to which the desired alphabet letter is assigned by the required number of times until the desired alphabet letter appears, referring to the relationship between keys and their assigned data given below.

To enter "T," for example, press the 1 key two times. At this stage, the "T" is highlighted but not established yet.

Keys	Key data assigned
7	A, B, C, a, b, c
8	D, E, F, d, e, f
9	G, H, I, g, h, i
4	J, K, L, j, k, l
5	M, N, O, m, n, o
6	P, Q, R, p, q, r
1	S, T, U, s, t, u
2	V, W, X, v, w, x
3	Y, Z, +, y, z
0	-, %, $, \
.	.comma (,), /, space

2) Press any of the following keys to establish the highlighted character ("T" in this example).

- If you press any one of the function keys (F1 to F8), BS, C, and magic keys (M1 to M4), then the highlighted character ("T") will be established. The key data of both the established key and the key you pressed now will be returned.
- If you press the ENT key, the highlighted character ("T") will be established and the key data will be returned.
- If you press the SF key, the alphabet entry mode will be switched to the numeric entry mode. The highlighted character will be ignored.
- If you press any other numerical key (e.g. "3" to which "Y" is assigned), the key data of the highlighted character ("T") will be established and the key data will be returned. At this state, the "Y" is not established yet.

When no key is ready to be established, pressing any of the function keys, BS, C, ENT, and magic keys will return the key data of the pressed key.

(Example: If you press the 1, 1, 2, and 3 keys)

The key data of "T" and "V" will be returned. The "Y" is not established yet.

(Example: If you press the C, 1, 1, 1, and ENT keys)

The 18H and "U" will be returned.

7.2.4 Other Facilities for the Keyboard

[1] Auto-repeat

The keys on the BHT series are not auto-repeat.

[2] Shift key

■ BHT-3000

The Shift key can be switched to non-lock type or lock type by selecting NRM or ONE on Set Resume screen in System Mode, respectively.

• Non-lock type	The keypad will be shifted only when the Shift key is held down.
• Lock type	Once the Shift key is pressed, the next one key pressed will be shifted and the following keys will not be shifted.

■ BHT-4000

The Shift key can be switched to non-lock type or lock type by selecting Non Lock or One Time on SET OTHERS menu in System Mode, respectively.

• Non-lock type	The keypad will be shifted only when the Shift key is held down.
• Lock type	Once the Shift key is pressed, the next one key pressed will be shifted and the following keys will not be shifted.

When the keys are shifted, the shift-key icon SF appears on the bottom line of the LCD if the system status indication is set to on. (You can turn on the system status indication on the SET DISPLAY menu in System Mode or by using the `OUT` statement.)

■ BHT-5000/BHT-6000/BHT-6500

The Shift key can be switched to non-lock type or lock type by selecting Nonlock or Onetime on shift key setting menu of the SET SYSTEM screen in System Mode, respectively.

• Non-lock type	The keypad will be shifted only when the Shift key is held down.
• Lock type	Once the Shift key is pressed, the next one key pressed will be shifted and the following keys will not be shifted.

When the keys are shifted, the shift-key icon SF appears at the right end of the bottom line of the LCD if the system status indication is set to on. (You can turn on the system status indication on the SET DISPLAY menu in System Mode or by using the `OUT` statement.)

■ BHT-7000/BHT-7500

The Shift key can be switched to non-lock type or lock type by selecting Nonlock or Onetime on the SET KEY menu in System Mode, respectively.

• Non-lock type	The keypad will be shifted only when the Shift key is held down.
• Lock type	Once the Shift key is pressed, the next one key pressed will be shifted and the following keys will not be shifted.

When the keys are shifted, a bar appears above the SF in the status display.

7.3 Timer and Beeper

7.3.1 Timer Functions

The timer functions (TIMEA, TIMEB, and TIMEC) are available in BHT-BASIC for accurate time measurement.

Use these timer functions for monitoring the keyboard waiting time, communications timeout errors, etc.

```
TIMEA = 100               ’10 sec
WAIT 0,&H10
BEEP
PRINT "10sec."

TIMEC = 20                ’2 sec
WAIT 0,&H41
BEEP
PRINT "2sec. or Keyboard"
```

7.3.2 BEEP Statement

The BEEP statement sounds a beeper and specifies the frequency of the beeper.

The example below sounds the musical scale of do, re, mi, fa, sol, la, si, and do.

```
READ readDat%
WHILE (readDat% >= 0)
    TIMEA = 3
    BEEP 2,,,readDat%
    WAIT 0,&h10
    READ readDat%
WEND
DATA 523,587,659,698,783,880,987,1046,-1
```

Specifying the frequency with value 0, 1, or 2 produces the special beeper effects; that is, the low-, medium-, or high-pitched tone, respectively.

```
FOR i% = 0 TO 2
    TIMEC = 20
    BEEP,,,i%
    WAIT 0,&h40
NEXT
```

NOTE Only if setting 0, 1, or 2 or making no specification to the frequency, you can adjust the beeper volume on the LCD when powering on the BHT. (For the adjustment of the beeper volume, refer to the BHT’s User’s Manual.)

7.4 Controlling and Monitoring the I/Os

7.4.1 Controlling by the `OUT` Statement

The `OUT` statement can control the input and output devices (I/Os) listed in Appendix D, I/O Ports." The table below lists some examples.

`OUT` Statement	I/O Devices

`OUT 1,&h02`
`OUT 1,&h01`
`OUT 1,&h00`
	Turns on the reading confirmation LED in green.
	Turns on the reading confirmation LED in red.
	Turns off the reading confirmation LED.

`OUT 3,&hXX (XX:` 00 to 07`)`	Sets the LCD contrast.

`OUT 4,&h00`
`OUT 4,&h01`
	Sets the Japanese message version.
	Sets the English message version.

`OUT 6,&hXX (XX:` 00 to FF`)`	Sets the sleep timer.

7.4.2 Monitoring by the `INP` Function

The `INP` function monitors the input and output devices (I/Os) listed in Appendix D, "I/O Ports." The table below lists some examples.

`INP` Function	I/O Devices	Value	Meaning
`INP(0) AND &h01`	Keyboard buffer status	1 0	Data present No data
`INP(0) AND &h02`	Bar-code buffer status	1 0	Data present No data
`INP(0) AND &h04`	Trigger switch status*	1 0	Being pressed Being released
`INP(0) AND &h08`	Receive buffer status	1 0	Data present No data
`INP(0) AND &h10`	`TIMEA` function	1	Set to 0
`INP(0) AND &h20`	`TIMEB` function	1	Set to 0
`INP(0) AND &h40`	`TIMEC` function	1	Set to 0

* In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, the `INP` function can monitor the trigger switch status only when the trigger switch function is assigned to any of the magic keys.

7.4.3 Monitoring by the `WAIT` Statement

The `WAIT` statement monitors the input and output devices (I/Os) listed in Appendix D, "I/O Ports." Unlike the `INP` function, the `WAIT` statement makes the I/O devices idle while no entry occurs, thus saving power consumption and increasing the battery service life.

The table below lists some examples.

`WAIT` Statement	I/O Devices
`WAIT 0,&h01`	Keyboard buffer status
`WAIT 0,&h02`	Bar-code buffer status
`WAIT 0,&h04`	Trigger switch status*
`WAIT 0,&h08`	Receive buffer status
`WAIT 0,&h10`	`TIMEA` function
`WAIT 0,&h20`	`TIMEB` function
`WAIT 0,&h40`	`TIMEC` function

* In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, the `WAIT` function can monitor the trigger switch status only when the trigger switch function is assigned to any of the magic keys.

In a single `WAIT` statement, you can specify more than one I/O device if the same port number applies. To monitor the keyboard buffer and the bar-code buffer with the single `WAIT` statement, for example, describe the program as shown below.

```
OPEN "BAR:" AS #10 CODE "A:"
WAIT 0,&h03
```

The above example sets the value of &h03 (00000011) to port 0, indicating that it keeps waiting until either bit 0 or bit 1 becomes ON by pressing any key or by reading a bar code.

Chapter 8
Files

CONTENS

8.1 File Overview

8.1.1 Data Files and Device I/O Files

BHT-BASIC treats not only data files but also bar code device I/Os and communications device I/Os as files, by assigning the specified names to them.

File Type	File Name	Remarks
Data File	`filename.extension`	
	`drivename:filename.extension`	(Applicable to the BHT-5000/BHT-6000/ BHT-6500/BHT-7000/ BHT-7500)
Device I/O File	`BAR:`	Bar code device
Device I/O File	`COM:`	Communications device

TIP Data files and user program files are stored in the user area of the memory.

8.1.2 Access Methods

To access data files or device I/O files, first use the `OPEN` statement to open those files. Input or output data to/from the opened files by issuing statements or functions to them according to their file numbers. Then, close those files by using the `CLOSE` statement.

8.2 Data Files

8.2.1 Overview

Like user programs, data files will be stored in the user area of the memory. The location of the user area differs depending upon the BHT series as shown below.

BHT series	Location of user area
BHT-3000/BHT-4000	A single drive (no drive specification)
BHT-5000/BHT-6000/BHT-6500	Drive A and drive B
BHT-7000/BHT-7500	Drive A and drive B*

* Drive B is provided for ensuring the compatibility with the BHT-5000/BHT-6000/BHT-6500.

The memory capacity available for data files differs depending upon BHT series as follows:

In the BHT-3000/BHT-4000, the memory space available for data files is (Memory space on the single drive - Memory space occupied by user programs). In the BHT-7000/BHT-7500, it is (Memory space on drive A - Memory space occupied by user programs). In the BHT-5000/ BHT-6000/BHT-6500, it is (Memory spaces on drives A and B - Memory space occupied by user programs).

For the memory mapping, refer to Appendix F, "Memory Area." You may check the current occupation of the memory with the `FRE` function.

8.2.2 Naming Files

The name of a data file generally contains *`filename.extension`*. The *`filename`* can have one to eight characters; the *`extension`* can have one to three characters.

In the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, the *`filename.extension`* should be preceded by the *`drivename`*. The *`drivename`* is A: or B:. If the *`drivename`* is omitted, the default A: applies.

The *`extension`* can be omitted. In such a case, a period should be also omitted. The following extensions cannot be used for data files:

Unavailable extensions for data files	`.PD3`, `.FN3`, `.EX3`, and `.FLD`

Programs make no distinction between uppercase and lowercase letters for drive names, file names, and extensions. They regard those letters as uppercase.

In the BHT-3000, the following file names cannot be used for data files since they are reserved for Easy Pack:

Reserved File Names
`PACK1.DAT`
`PACK2.DAT`
`PACK3.DAT`
`PACK4.DAT`

8.2.3 Structure of Data Files

■ Record

A data file is made up of a maximum of 32767 records. A record is a set of data in a data file and its format is defined by the `FIELD` statement. The maximum length of a record is 255 bytes including the number of the character count bytes* (= the number of the fields).

* When transferring data files, the BHT-protocol/BHT-Ir protocol automatically prefixes a character count byte in binary format to each data field.

■ Field

A record is made up of 1 to 16 fields. Data within the fields will be treated as character (ASCII) data.

Each field precedes a character count byte in binary format, as described above. Including that one byte, the maximum length of a field is 255 bytes.

The following `FIELD` statement defines a record which occupies a 28-byte memory area (13 + 5 + 10 bytes) for data and a 3-byte memory area for three character count bytes. Totally, this record occupies not a 28-byte area but a 31-byte area in the memory.

```
FIELD #2,13 AS bardat$,5 AS keydat$,10 AS dt$
'1+13+1+5+1+10=31 bytes
```

* When a data file is transmitted according to the BHT-protocol, the following conditions should be also satisfied:

- The maximum length of a field is 254 bytes (99 bytes in the BHT-3000/BHT-4000) excluding a character count byte.

8.2.4 Data File Management by Directory Information

The Interpreter manages data files using the directory information stored in the system area of the memory.

The directory information, for example, contains the following:

```
filename.extension
Information of Each Field (Field length)
Number of Written Records
Maximum Number of Registrable Records
```

- **Number of Written Records**

  Means the number of records already written in a data file, which the `LOF` function can return.

  If no record number is specified in the `PUT` statement, the Interpreter automatically assigns a number of (the current written record number + 1) to the record.

  ```
  PUT #1
  ```

- **Maximum Number of Registrable Records**

  You may declare the maximum number of records registrable in a data file by using the `RECORD` option in the `OPEN` statement, as shown below.

```
        OPEN "work.DAT" AS #10 RECORD 50
        FIELD #10,13 AS code$,5 AS price$
```

  The above program allows you to write up to 50 records in the data file named `work`.DAT.

  If the statement below is executed following the above program, a run-time error will occur.

```
        PUT #10,51
```

  The maximum number of registrable records can be optionally specified only when you make a new data file. If designated to the already existing data file, the specification will be ignored without occurrence of a run-time error.

  If the BHT-7000/BHT-7500 receives a file with the `XFILE` statement, it will automatically set the maximum number of registrable records to 32,767 for that file.

  Other BHT series will make such setting only when it receives a file not existing in the BHT with the `XFILE` statement.

  Specifying the maximum number of registrable records will not cause the Interpreter to reserve the memory area.

8.2.5 Programming for Data Files

■ Input/Output for Numeric Data

- To write numeric data into a data file:

It is necessary to use the STR$ function for converting the value of a numeric expression into a string.

To write -12.56 into a data file, for example, the field length of at least 6 bytes is required. When using the FIELD statement, designate the sufficient field length; otherwise, the data will be lost from the lowest digit when written to the field.

- To read data to be treated as a numeric from a data file:

Use the VAL function for converting a string into a numeric value.

■ Data Retrieval

The SEARCH function not only helps you make programs for data retrieval efficiently but also makes the retrieval speed higher.

The SEARCH function searches a designated data file for specified data, and returns the record number where the search data is first encountered. If none of the specified data is encountered, this function returns the value 0.

■ Deletion of Data Files

The CLFILE or KILL statement deletes the designated data file.

CLFILE	Erases only the data stored in a data file without erasing its directory information, and resets the number of written records to 0 (zero) in the directory. This statement is valid only to opened data files.
KILL	Deletes the data stored in a data file together with its directory information. This statement is valid only to closed data files.

• Program sample with the CLFILE statement

```
OPEN "work2.DAT" AS #1
FIELD #1,1 AS a$
CLFILE #1
CLOSE #1
```

• Program sample with the KILL statement

```
CLOSE
KILL "work2.DAT"
```

■ Restrictions on Input/Output of Data Files

No `INPUT#`, `LINE INPUT#`, or `PRINT#` statement or `INPUT$` function can access data files. To access data files, use a `PUT` or `GET` statement.

■ Drive Defragmentation (BHT-7000/BHT-7500 only)

During downloading in the BHT-7000/BHT-7500, a delay of a few seconds (response delay from the BHT) may occur according to the user area condition.

To eliminate the delay, defragment the drive for the size required for downloading beforehand. Doing so will also reduce the device open time in communications. Defragmentation before downloading is recommended.

If there is no specified size of the empty area in the drive, it is necessary to defragment the whole empty area.

In complicated write operation, any of the following symptoms may be caused in units of a few seconds. If such occurs frequently, defragment the drive.

- The beeper sound is prolonged.
- Keys do not work.
- No bar code entry is possible.
- Switching the LCD screen is delayed.
- No data can be received.
- Timeout by the `TIMEA/TIMEB/TIMEC` is delayed.

The `OUT` statement may defragment the drive. In the `OUT` statement, you may specify the size of the empty area to be defragmented in units of 4 kilobytes, starting with 4 kilobytes up to the maximum size of the user area.

During drive defragmentation, user programs will be halted. Upon completion of defragmentation, they will resume operation.

In the `OUT` statement, you may also select whether a bar graph showing the progress of defragmentation will be displayed on the LCD. The bar graph, if selected, will disappear after completion of defragmentation and the previous screen will come back.

For details about defragmentation with `OUT` statement, refer to Appendix D, "I/O Ports," D5.

8.2.6 About Drives

The BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500 has logical drives.

■ BHT-5000/BHT-6000/BHT-6500

Drive A and drive B are defined on the RAM and flash ROM, respectively. Accordingly, the file access is partially different between drive A and drive B as listed below.

File access operation	To drive A	To drive B
Download	`XFILE` statement	Same as left.
Create	New with `OPEN` statement	Run-time error (43h)
Open	Open with `OPEN` statement	Same as left.
Read	`GET` statement	Same as left.
Write	`PUT` statement	Run-time error (43h)
Close	`CLOSE` statement	Same as left.
Clear	`CLFILE` statement	Run-time error (43h)
Delete	`KILL` statement	Same as left.*

* The BHT deletes data actually when next downloading takes place. To delete data, the BHT-5000/BHT-6000 (System version 2.00 or later)/BHT-6500 uses an empty area of drive A by 64 kilobytes. The BHT-6000 (System version earlier than 2.00) uses it by 128 kilobytes. If there is no such space in drive A, a run-time error (44h) will occur.

■ BHT-7000/BHT-7500

Drive B is provided for ensuring compatibility with the BHT-5000/BHT-6000/BHT-6500.

If you specify drive name "B:" preceding a filename.extension and open an existing file, the BHT will open the file as a read-only file. Executing the `PUT` statement to the read-only file, a run-time error (43h) will result.

If you specify drive name "A:" or omit a drive name, the BHT will open the file as a read/write file.

The `XFILE` and `KILL` statements will ignore drive names "A:" and "B:."

The table below lists the file access details relating to drives.

File access operation	To drive A	To drive B
Download	`XFILE` statement	Same as left.
Create	New with `OPEN` statement	Run-time error (43h)
Open	Open with `OPEN` statement	Same as left.
Read	`GET` statement	Same as left.
Write	`PUT` statement	Run-time error (43h)
Close	`CLOSE` statement	Same as left.
Clear	`CLFILE` statement	Run-time error (43h)
Delete	`KILL` statement	Same as left.

8.3 Bar Code Device

8.3.1 Overview

■ Opening the Bar Code Device by `OPEN "BAR:"` Statement

The `OPEN "BAR:"` statement opens the bar code device. In this statement, you may specify the following bar code types available in the BHT. The BHT can handle one of them or their combination.

Available Bar Code Types		Default Settings
Universal product codes	EAN-13* EAN-8 UPC-A* UPC-E	No national flag specified.
Interleaved 2 of 5 (ITF)		No read data length specified. No check digit.
Standard 2 of 5 (STF)**		No read data length specified. No check digit. Short format of the start/stop characters supported.
Codabar (NW-7)		No read data length specified. No check digit. No start/stop character.
Code 39		No read data length specified. No check digit.
Code 93		No read data length specified.
Code 128***		No read data length specified.

* Reading wide bars

The EAN-13 and UPC-A bar codes may be wider than the readable area of the bar-code reading window.

BHT-3000: Such wider bars can be read by the double-touch reading feature. Read first the right (or left) half of the bar code together with the center bar and then read the remaining half. The system combines the split data into one bar code. For activation/ deactivation of the double-touch reading feature, refer to the "BHT-3000 User's Manual."

BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500: Such wider bars can be read by long-distance scanning. Pull the bar-code reading window away from the bar code so that the entire bar code comes into the illumination range. (No double-touch reading feature is supported.)

** The STF can be read by the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500.

*** In the BHT-5000/BHT-6500/BHT-7000/BHT-7500, specifying the Code 128 makes it possible to read not only the Code 128 but also the EAN-128.

■ Specifying Options in the `OPEN "BAR:"` Statement

You may also specify several options as listed below for each of the bar code types in the `OPEN "BAR:"` statement.

Options
- Check digit (only for ITF, Codabar, Code 39, and STF) - Read data length - Start/stop character (only for Codabar and STF) - Start character flag (only for universal product codes) - Supplemental code (only for universal product codes. Not supported by the BHT-3000)

■ Bar Code Buffer

The bar code buffer stores the inputted bar code data. It will be occupied by one operator entry job and can contain up to 40 characters in the BHT-3000 and 99 characters in the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500.

You can check whether the bar code buffer stores bar code data, by using any of the `EOF`, `INP`, and `LOC` functions, and the `WAIT` statement.

Any of the `INPUT#` and `LINE INPUT#` statements, and the `INPUT$` function reads bar code data stored in the buffer into a string variable.

8.3.2 Programming for Bar Code Device

■ Code Mark

The `MARK$` function allows you to check the code type and the length of the inputted bar code data.

This function returns a total of three bytes: one byte for the code mark (denoting the code type) and two bytes for the data length.

■ Multiple Code Reading

You may activate the multiple code reading feature which reads more than one bar code type while automatically identifying them, by designating the desired bar code types following the `CODE` in the `OPEN "BAR:"` statement.

■ Read Mode of the Trigger Switch

In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, the trigger switch function is assigned to the following magic keys by default:

BHT-6000:	M1 and M2 keys
BHT-6500:	M1, M2, M3, and M4 keys
BHT-7000/BHT-7500:	M3 and M4 keys

In the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, you may assign the trigger switch function to other keys by using the `KEY` statement.

You may select the read mode of the trigger switch by using the `OPEN "BAR:"` statement.

Read Mode	`OPEN "BAR:"` Statement
Auto-off Mode (Default)	`OPEN "BAR:F"`...
Momentary Switching Mode	`OPEN "BAR:M"`...
Alternate Switching Mode	`OPEN "BAR:A"`...
Continuous Reading Mode	`OPEN "BAR:C"`...

To check whether the trigger switch is pressed or not, use the `INP` function or the `WAIT` statement, as shown below.

```
trig% = INP(0) AND &h04
```

If the value of the `trig%` is 04h, the trigger switch is kept pressed; if 00h, it is released.

■ Generation of Check Digit

Specifying a check digit in the `OPEN "BAR:"` statement makes the Interpreter automatically check bar codes. If necessary, you may use the `CHKDGT$` function for generating a check digit of bar code data.

■ Controlling the Reading Confirmation LED and Beeper (Vibrator) at the Time of Scanning for Confirmation of Successful Reading (BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500)

By using the `OPEN "BAR:"` statement, you can control:

- whether the reading confirmation LED should light in green or not (Default: Light in green)
- whether the beeper should beep or not (Default: No beep)
  The BHT-6500/BHT-7000/BHT-7500 may control the vibrator also.

when a bar code is read successfully.

Controlling the reading confirmation LED

If you have activated the reading confirmation LED (in green) in the `OPEN "BAR:"` statement, the `OUT` statement cannot control the LED via output port 1 when the bar code device file is opened. (For details about settings of bits 0 and 1 on output port 1, refer to Appendix D.)

If you have deactivated the reading confirmation LED in the `OPEN "BAR:"` statement, the `OUT` statement can control the LED via output port 1 even when the bar code device file is opened. (For details about settings of bits 0 and 1 on output port 1, refer to Appendix D.)

This way, you can control the reading confirmation LED, enabling that:

- a user program can check the value of a scanned bar code and turn on the green LED when the bar code has been read successfully.
  (For example, you can make the user program interpret bar code data valued from 0 to 100 as correct data.)
- a user program can turn on the red LED the moment the bar code has been read.

Controlling the beeper (vibrator)

If you activate the beeper in the `OPEN "BAR:"` statement, the BHT will beep when it reads a bar code successfully.

In the BHT-6500/BHT-7000/BHT-7500, you may choose beeping only, vibrating only, or beeping & vibrating on the LCD screen or by setting the output port in the `OUT` statement.

8.4 Communications Device

8.4.1 Hardware Required for Data Communications

[1] BHT-3000/BHT-4000/BHT-5000

The following hardware is required for communications between the BHT and the host computer:

- Optical communications unit (CU-3000/CU-4000/CU-5000) and its interface cable

or

- Direct-connect interface cable

For the communications specifications, refer to the "BHT's User's Manual."

[2] BHT-6000/BHT-6500/BHT-7000/BHT-7500

The following hardware is required for communications between the BHT and the host computer:

- Optical communications unit (CU-6000/CU-7000) and its interface cable

or

- Direct-connect interface cable

For the communications specifications, refer to the "BHT's User's Manual."

Using Ir-Transfer Utility E allows the BHT to directly communicate with the IR port-integrated host computer or an external IR transceiver. For details about IR port-integrated computers and external IR transceivers available, refer to the "Ir-Transfer Utility E Guide."

8.4.2 Programming for Data Communications

Setting the Communications Parameters

Use the `OPEN "COM:"` statement to set the communications parameters.

[1] BHT-3000/BHT-4000/BHT-5000

Communications Parameters	Effective Setting	Default
Transmission speed (bps)	HS[*1], 38400[*2], 19200, 9600, 4800, 2400, 1200, 600, or 300	9600
Parity	None, even, or odd	None
Character length	7 or 8 bits	8 bits
Stop bit length	1 or 2 bits	1 bit
RS/CS control	Yes or no	No
Timeout detection	Yes or no	No
RS control[*3]	ON (1) or OFF (0)	ON (1)
ER control[*4]	ON (1) or OFF (0)	ON (1)

[*1] The HS (High Speed) is available only in file transmission between the BHT-4000 and host computer by using Transfer Utility. It requires the RS-232C interface specially connected. Refer to the "BHT-4000 User’s Manual."

[*2] The 38400 bps is available in the BHT-3000/BHT-4000/BHT-5000. Note that in the BHT-3000/BHT-4000, the direct-connect interface port should be selected.

[*3] The RS control is supported in the BHT-4000/BHT-5000.

[*4] The ER control is supported when the direct-connect interface is selected in the BHT-4000.

[2] BHT-6000/BHT-6500/BHT-7000/BHT-7500

■ For optical interface

Communications Parameters	Effective Setting	Default
Transmission speed (bps)	115200, 57600, 38400, 19200, 9600, or 2400	9600

Parameters other than the transmission speed are fixed (Character length = 8 bits, Parity = None, Stop bit length = 1 bit), since the physical layer of the optical interface complies with the IrDA-SIR 1.0.

■ For direct-connect interface

Communications Parameters	Effective Setting	Default
Transmission speed (bps)	115200[*1], 57600[*1], 38400, 19200, 9600, 4800, 2400, 1200, 600, or 300	9600
Parity[*2]	None, even, or odd	None
Character length[*2]	7 or 8 bits	8 bits
Stop bit length[*2]	1 or 2 bits	1 bit

[*1] The 115200 bps and 57600 bps are available in the BHT-7000/BHT-7500.

[*2] The parity, character length, and stop bit length are fixed to none, 8 bits, and 1 bit, respectively, if the BHT-Ir protocol is selected.

8.4.3 Overview of Communications Protocols

The BHT series supports the three communications protocols—BHT-protocol, BHT-Ir protocol, and multilink protocol for file transmission, as listed below. Using the `XFILE` statement, the BHT may upload or download a file according to any of these protocols.

• BHT-protocol	:	All BHT series
• BHT-Ir protocol	:	BHT-6000/BHT-6500/BHT-7000/BHT-7500
• Multilink protocol	:	BHT-5000

[1] BHT-protocol

All BHT series supports the BHT-protocol.

This protocol is used also in System Mode or Easy Pack.

For the communications specifications of the BHT-protocol, refer to the BHT User's Manual.

■ Primary station and secondary station

The primary station and the secondary station should be defined as below.

- When uploading data files

Primary station:	BHT
Secondary station:	Host computer

- When downloading data files

Primary station:	Host computer
Secondary station:	BHT

■ Protocol functions

In the BHT-protocol, using the following protocol functions may modify a transmission header or terminator in a send data:

For a header:	`SOH$` or `STX$`
For a terminator:	`ETX$`

■ Field length that the BHT-protocol can handle

When the BHT-3000/BHT-4000 transmits files according to the BHT-protocol, each field length should be a maximum of 99 bytes. The BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500 may transmit files having the field length of up to 254 bytes.

In file transmission, the host computer should also support the same field length as the BHT. The MS-DOS–based Transfer Utility supports the field length of up to 99 bytes; the Windows-based Transfer Utility supports up to 254 bytes.

[2] BHT-Ir protocol (BHT-6000/BHT-6500/BHT-7000/BHT-7500)

In addition to the BHT-protocol, the BHT-6000/BHT-6500/BHT-7000/BHT-7500 supports the BHT-Ir protocol.

If you select the BHT-Ir protocol by using the `OUT` statement (Port No. &h6060) or in System Mode, you can upload or download a data file with the `XFILE` statement.

The BHT-Ir protocol is used also in System Mode or Easy Pack Pro.

For the communications specifications of the BHT-Ir protocol, refer to the "BHT-6000 User's Manual," "BHT-6500 User's Manual," "BHT-7000 User's Manual," or "BHT-7500 User's Manual."

■ Primary station and secondary station

The primary station and the secondary station should be defined as below.

- When uploading data files

Primary station:	BHT-6000/BHT-6500/BHT-7000/BHT-7500
Secondary station:	Host computer

- When downloading data files

Primary station:	Host computer
Secondary station:	BHT-6000/BHT-6500/BHT-7000/BHT-7500

■ Protocol functions

In the BHT-Ir protocol, you cannot change the values of the headers and terminator with the protocol functions in BHT-BASIC.

[3] Multilink protocol (BHT-5000 only)

In addition to the BHT-protocol, the BHT-5000 may support the multilink protocol which is used for file transmission between the host computer and more than one BHT-5000 (placed on the multilinked CU-5003s), provided that Multilink Protocol System (MLTU3.EX3) is downloaded to the BHT-5000 beforehand.

To transfer files by using the multilink protocol, you need Multilink Transfer Utility (MLTU3.EXE) to be run in the host computer and the CU-5003(s). For details, contact your nearest dealer.

If you download the Multilink Protocol System to the BHT-5000 and select the multilink protocol by using the `OUT` statement (Port No. &h6060) or in System Mode, then you can upload or download files according to the multilink protocol with the `XFILE` statement.

The multilink protocol is used also in System Mode.

■ Master station and slave station

The master station and the slave station should be defined as below.

Master station:	Host computer
Slave station:	BHT-5000

■ Primary station and secondary station

The primary station and the secondary station should be defined as below.

- When uploading data files

Primary station:	BHT-5000
Secondary station:	Host computer

- When downloading data files

Primary station:	Host computer
Secondary station:	BHT-5000

■ Protocol functions

In the multilink protocol, you cannot change the values of the headers and terminator with the protocol functions in BHT-BASIC.

8.4.4 File Transfer Tools

[1] Transfer Utility

Transfer Utility is optionally available in two versions: MS-DOS–based and Windows-based. It supports the BHT-protocol and allows you to upload or download user program files and data files between the host and the BHT, when invoked by the `XFILE` statement.

This utility can also transfer user program files and data files to/from System Mode.

NOTE If you have modified transmission headers or terminator to any other character codes by using the protocol functions, Transfer Utility is no longer available.

For computers and Windows version which are available for Transfer Utility and the operating procedure of Transfer Utility, refer to the "Transfer Utility Guide."

[2] Ir-Transfer Utility C

Ir-Transfer Utility C is optionally available in two versions: MS-DOS–based and Windows-based. It supports the BHT-Ir protocol and allows you to upload or download user program files and data files between the host and the BHT-6000/BHT-6500/BHT-7000/BHT-7500, when invoked by the `XFILE` statement. Ir-Transfer Utility C handles IrDA SIR-compliant communications via the communications unit CU.

This utility can also transfer user program files and data files to/from System Mode.

For computers and Windows versions which are available for Ir-Transfer Utility C and the operating procedure of Ir-Transfer Utility C, refer to the "Ir-Transfer Utility C Guide."

[3] Ir-Transfer Utility E

Ir-Transfer Utility E is optional Windows-based software. It supports the BHT-Ir protocol and allows you to upload or download user program files and data files between the host and the BHT-6000/BHT-6500/BHT-7000/BHT-7500, when invoked by the `XFILE` statement. Ir-Transfer Utility E handles IrDA SIR-compliant communications via the IR port integrated in a computer or an external IR transceiver.

This utility can also transfer user program files and data files to/from System Mode.

For computers and Windows versions which are available for Ir-Transfer Utility E and the operating procedure of Ir-Transfer Utility E, refer to the "Ir-Transfer Utility E Guide."

[4] Multilink Transfer Utility (BHT-5000 only)

Multilink Transfer Utility is optional MS-DOS–based software. It supports the multilink protocol and allows you to upload or download user program files and data files between the host and the BHT-5000 (placed on the multilinked CU-5003s), when invoked by the `XFILE` statement.

This utility can also transfer user program files and data files to/from System Mode.

For computers available for Multilink Transfer Utility and the operating procedure of Multilink Transfer Utility, refer to the "Multilink Transfer Utility Guide."

Chapter 9
Event Polling and Error/Event Trapping

CONTENTS

9.1 Overview

BHT-BASIC supports event polling and two types of trapping: error trapping and event trapping.

- Event polling
- Trapping
  - Error trapping
  - Event (of keystroke) trapping

■ Event Polling

Makes programs monitor the input devices for occurrence of events.

■ Error Trapping

Traps a run-time error and handles it by interrupt to transfer control to the error-handling routine.

If a run-time error occurs when this trapping ability is disabled, the Interpreter will terminate the current user program while showing the error message.

■ Event (of Keystroke) Trapping

Traps a particular keystroke (caused by pressing any of the specified function keys) and handle it by interrupt to transfer control to the event-handling routine.

9.2 Event Polling

[1] Programming sample

The program below shows the event polling example which monitors the bar code reader and the keyboard for occurrence of events.

This example uses the `EOF` and `INKEY$` functions to check the data input for the bar code reader and the keyboard, respectively.

```
            OPEN "BAR:" AS #1 CODE "A"
loop
            WAIT 0,3
            IF NOT EOF(1) THEN
                GOSUB barcod
            ENDIF
            k$=INKEY$
            IF k$<>"" THEN
                GOSUB keyin
            ENDIF
            GOTO loop
barcod
            BEEP
            LINE INPUT #1,dat$
            PRINT dat$
            RETURN
keyin
              ⋮
            RETURN
```

[2] I/O devices capable of being monitored by the event polling

Listed below are the I/O devices which the event polling can monitor.

I/O Devices	Monitor Means	Events
Keyboard	`INKEY$` function	Input of one character from the keyboard
Bar code reader	`EOF` or `LOC` function	Presence/absence of bar code data input or the number of read characters (bytes)
Receive buffer	`EOF`, `LOC`, or `LOF` function	Presence/absence of receive data or the number of received characters (bytes)
Timer	`TIMEA`, `TIMEB`, or `TIMEC` function	Timer count-up

■ Monitoring with the `INP` Function

Combining the `INP` function with the above functions enables more elaborate programming for event polling.

For the `INP` function, refer to Appendix D, "I/O Ports."

9.3 Error Trapping

[1] Overview

If a run-time error occurs during program running, error trapping makes the program cause an interrupt upon completion of the machine statement so as to transfer control from the current program to the error-handling routine which has been specified by a label.

If a run-time error occurs when this trapping ability is disabled, the Interpreter will terminate the current user program while displaying the error message as shown below.

Error message sample:

```
ERL=38A4 ERR=34
```

The above message indicates that a run-time error has occurred at address 38A4h and its error code is 34h. Both the address and error code are expressed in hexadecimal notation.

The address is a relative address and corresponds to the address in the program list outputted by the Compiler. According to this address indication, you can pinpoint the program line where the run-time error has occurred.

The error code 34h (52 in decimal notation) means that the user program attempted to access a file not opened. (Refer to Appendix A1, “Run-time Errors.")

The `ERL` and `ERR` functions described in an error-handling routine will return the same values, 38A4h and 34h, respectively.

NOTE If an error occurs during execution of user-defined functions or sub routines so that the error is trapped and handled by the error-handling routine, then do not directly pass control back to the main routine having the different stack level by using the `RESUME` statement. The return address from the user-defined functions or subroutines will be left on the stack, causing a run-time error due to stack overflow.

To prevent such a problem, once transfer control to the routine which caused the interrupt in order to match the stack level and then jump to any other desired routine. (Refer to Chapter 3, Section 3.1, "Program Overview.")

[2] Programming for trapping errors

To trap errors, use the ON ERROR GOTO statement in which you should designate the error-handling routine (to which control is to be transferred if a run-time error occurs) by the label.

```
                ON ERROR GOTO err01
                    ⋮
                (Main routine)
                    ⋮
                END
     err01
                (Error-handling routine)
                PRINT"*** error ***"
                PRINTERR,HEX$(ERL)
                RESUME NEXT
```

If a run-time error occurs in the main routine, the above program executes the error-handling routine specified by label err01 in the ON ERROR GOTO statement.

In the error-handling routine, the ERL and ERR functions allow you to pinpoint the address where the error has occurred and the error code, respectively.

NOTE According to the error location and error code, you should troubleshoot the programming error and correct it for proper error handling.

The RESUME statement may pass control from the error-handling routine back to any specified statement as listed below.

RESUME Statement	Description
RESUME or RESUME 0	Resumes program execution with the statement that caused the error.
RESUME NEXT	Resumes program execution with the statement immediately following the one that caused the error.
RESUME *label*	Resumes program execution with the statement designated by *label*.

9.4 Event (of Keystroke) Trapping

[1] Overview

If any of the function keys previously specified for keystroke trapping is pressed, event trapping makes the program cause an interrupt so as to transfer control from the current program to the specified event-handling routine.

This trapping facility checks whether any of the function keys is pressed or not between every execution of the statements.

[2] Programming for trapping keystrokes

To trap keystrokes, use both the ON KEY...GOSUB and KEY ON statements. The ON KEY...GOSUB statement designates the key number of the function key to be trapped and the event-handling routine (to which control is to be transferred if a specified function key is pressed) in its label. The KEY ON statement activates the designated function key.

This trapping cannot take effect until both the ON KEY...GOSUB and KEY ON statements have been executed.

The keystroke of an unspecified function key or any of the numerical keys cannot be trapped.

The following program sample will trap keystroke of function keys F1, F2, and F3 (these keys are numbered 1, 2, and 3, respectively).

```
          ON KEY (1) GOSUB sub1
          ON KEY (2) GOSUB sub2
          ON KEY (3) GOSUB sub3
          KEY (1) ON
          KEY (2) ON
          KEY (3) ON
                ⋮
          (Main routine)
                ⋮
          END
sub1
          (Event-handling routine 1)
          RETURN
sub2
          (Event-handling routine 2)
          RETURN
sub3
          (Event-handling routine 3)
          RETURN
```

The `RETURN` statement in the event-handling routine will return control to the statement immediately following that statement where the keyboard interrupt occurred.

Even if a function key is assigned a null string by the `KEY` statement, pressing the function key will cause a keyboard interrupt when the `KEY ON` statement activates that function key.

If function keys specified for keystroke trapping are pressed during execution of the following statements or functions relating keyboard input, this trapping facility operates as described below.

Statements or Functions	Keystroke Trapping
`INPUT` statement	Ignores the entry of the pressed key and causes no interrupt.
`LINE INPUT` statement	Same as above.
`INPUT$` function	Same as above.
`INKEY$` function	Ignores the entry of the pressed key, but causes an interrupt.

Chapter 10
Sleep Function

CONTENTS

10.1 Sleep Function

The BHT supports the sleep function that automatically interrupts program execution if no event takes place within the specified length of time in the BHT, thereby minimizing its power consumption. Upon detection of any event, the BHT in the sleep state immediately starts the interrupted user program.

By using the `OUT` statement, you may set the desired length of time to the sleep timer within the range from 0 to 25.5 seconds in increment of 100 ms. The default is 1 second.

When setting the sleep timer, the `OUT` statement also copies (assigns) the set value to its internal variable. The sleep timer immediately starts counting down the value assigned to the internal variable, -1 per 100 ms. If the value becomes 0, the BHT goes into a sleep.

Note that the sleep time will not count in any of the following cases. When the BHT exits from any of them, the value preset to the sleep timer will be assigned to the internal variable again and the sleep timer will start counting.

- While a communications device file is opened by an `OPEN "COM:"` statement.
- During execution of a `SEARCH`, `DATE$`, or `TIME$` function
- When a `TIMEA`, `TIMEB`, or `TIMEC` function returns a nonzero value.
- When the bar code device file is opened by the `OPEN "BAR:"` statement under any of the following conditions:
  - With the continuous reading mode specified
  - With the momentary switching mode or auto-off mode specified, and with the trigger switch held down
  - With the alternate switching mode, and with the illumination LED (laser beam in the BHT-6500/BHT-7500) being on
- When any key is held down.
- When the LCD backlight is on.
- When the beeper is beeping.
- When the vibrator is working. (BHT-6500/BHT-7000/BHT-7500 only)
- When the BHT is updating data on the screen.
- When the BHT is writing data into a data file.
- When a register variable is undergoing change.

Chapter 11
Resume Function

CONTENTS

11.1 Resume Function

The resume function automatically preserves the current status of a running application program (user program or Easy Pack) when the BHT is powered off, and then resumes it when the BHT is powered on. That is, even if you unintentionally turn off the BHT or the automatic powering-off function turns off the BHT, turning on the BHT once again resumes the previous status of the program to allow you to continue the program execution.

The resume function is effective also during data transmission in execution of an application program, but a few bytes of data being transmitted may not be assured.

NOTE Even if you become disoriented with the operation during execution of an application program so as to power off the BHT when the resume function is enabled, the BHT cannot escape you from the current status of the program. This is because the resume function will not initialize the variables or restart the BHT. (You can disable the resume function in System Mode.)

The resume function does not work after execution of System Mode or any of the following commands:

- `END`
- `POWER OFF`
- `POWER 0`

NOTE In preparation for maintenance or inspection jobs involving execution of System Mode (which will disable the resume function), store important information contained in user programs by using files or register variables, preventing your current operation jobs from getting crippled.

Chapter 12
Power-related Functions

CONTENTS

12.1 Low Battery Warning

■ BHT-3000

If the battery voltage is below the specified level when the BHT-3000 is powered on, the "Battery voltage has lowered" message appears on the LCD.

If the battery voltage drops while the BHT-3000 is in operation, the beeper beeps three times every 10 seconds.

If you keep using the BHT-3000 without battery replacement after the above warning, the BHT-3000 displays the "Replace the batteries" message on the LCD and turns itself off automatically.

Refer to the "BHT-3000 User’s Manual."

■ BHT-4000

If the output voltage of the Ni-Cd battery cartridge or dry battery cartridge drops below the specified level, the BHT-4000 displays the "Charge the battery!!" message, beeps five times, and then turns itself off automatically.

Refer to the "BHT-4000 User’s Manual."

■ BHT-5000

If the output voltage of the Ni-MH battery cartridge or dry battery cartridge drops below the specified level, the BHT-5000 displays the "Charge the battery!" message or "Replace the batteries" message, respectively, beeps five times, and then turns itself off automatically.

Refer to the "BHT-5000 User’s Manual."

■ BHT-6000/BHT-6500

If the output voltage of the battery cartridge drops below a specified lower level limit when the BHT-6000/BHT-6500 is in operation, the BHT displays the Level-1 message "Battery voltage has lowered." on the LCD and beeps three times. After that, it will resume previous regular operation.

If the battery output voltage drops further, the BHT-6000/BHT-6500 displays the Level-2 message "Charge the battery!" or "Replace the batteries!" (when driven by Ni-MH battery cartridge or dry batteries, respectively), beeps five times, and then turns itself off automatically.

Refer to the "BHT-6000 User’s Manual" or "BHT-6500 User’s Manual."

■ BHT-7000/BHT-7500

If the output voltage of the battery cartridge drops below a specified lower level limit when the BHT-7000/BHT-7500 is in operation, the BHT displays the Level-1 message "Battery voltage has lowered." on the LCD and beeps three times. After that, it will resume previous regular operation.

If the battery output voltage drops further, the BHT-7000/BHT-7500 displays the Level-2 message "Charge the battery!" or "Replace the batteries!" (when driven by the lithium-ion battery cartridge or dry battery cartridge, respectively), beeps five times, and then turns itself off automatically.

Refer to the "BHT-7000 User’s Manual" or "BHT-7500 User’s Manual."

12.2 Prohibited Simultaneous Operation of the Beeper*, Illumination LED (Laser Source**), and LCD Backlight

(* Beeper and vibrator in the BHT-6500/BHT-7000/BHT-7500)
(** Laser source in the BHT-6500/BHT-7500)

■ BHT-3000

The BHT-3000 is so designed that the beeper, illumination LED, and LCD backlight will not work simultaneously to save power consumption at peak load. There are priority orders among them; that is, the beeper has the highest priority, the illumination LED has the next priority, and the LCD backlight has the lowest priority. To beep the beeper when the LCD backlight is on, for example, the BHT-3000 turns off the LCD backlight once and then beeps.

■ BHT-5000

The BHT-5000 is so designed that the beeper and illumination LED will not work simultaneously to save power consumption at peak load. There is a priority order between them; that is, the beeper has the priority over the illumination LED. To beep the beeper at the time of bar code scanning, for example, the BHT-5000 turns off the illumination LED when beeping.

■ BHT-6000

The BHT-6000 is so designed that the illumination LED and the LCD backlight will not work simultaneously to save power consumption at peak load. There is a priority order between them; that is, the illumination LED has the priority over the LCD backlight.

■ BHT-6500/BHT-7000/BHT-7500

The BHT-6500/BHT-7000/BHT-7500 is so designed that the beeper (and vibrator), illumination LED (laser source in the BHT-6500/BHT-7500), and LCD backlight will not work simultaneously to save power consumption at peak load. There are priority orders among them; that is, the beeper (and vibrator) has the highest priority, the illumination LED (laser source) has the next priority, and the LCD backlight has the lowest priority.

12.3 Wakeup Function

■ BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500

The wakeup function allows you to turn on the BHT from "OFF" at the wakeup time (of the system clock) specified in user programs.

To set the wakeup time by using the `TIME$` function:

(1)	Set 1 to bit 2 on port 8.	Switches the `TIME$` function to the setting of the wakeup time.
(2)	Set the wakeup time by using the `TIME$` function.	
(3)	Set 1 to bit 0 on port 8.	Activates the wakeup function.

To confirm the wakeup time preset:

(1)	Set 1 to bit 2 on port 8.	Switches the `TIME$` function to the setting of the wakeup time.
(2)	Retrieve the wakeup time by using the `TIME$` function.	

NOTE

If you set or retrieve the system time or wakeup time by using the TIME$ function, the value of bit 2 on port 8 will be automatically reset to zero.

When bit 2 on port 8 is zero, you can set or retrieve the current system time by using the TIME$ function.

By reading the value of bit 1 on port 8 in user programs, you may confirm the initiation option of the BHT. If this bit is 1, the BHT is initiated by the wakeup function and if 0, it is initiated by the PW key.

12.4 Remote Wakeup Function (BHT-7000/BHT-7500)

[1] Outline

The remote wakeup function allows you to remotely power on the BHT and run the specified user program (hereafter referred to "remote wakeup program") by sending the specified message from the host computer to the BHT via the CU.

Developing user programs utilizing the remote wakeup at both the host computer and BHT enables you to automatically maintain the master system or update user programs.

To use the remote wakeup between the BHT and host computer, the following is required:

- Optical communications unit CU-7001 (The CU-7002 does not support the remote wakeup.)
- CU interface cable

[2] Remote wakeup operation

■ At the BHT

(1) Power off the BHT and put it on the CU.

The BHT will enter into the charge state* (i.e., into sleep). (For details about charging, refer to the "BHT-7000 User's Manual" or "BHT-7500 User's Manual.")

*Charge state: Charging or charged-up state

(2) Upon receipt of any data via the IR port, the BHT wakes up and becomes ready to receive data.

If no data comes in within the specified time (described in [3], ■ Setting the remote wakeup), the BHT will go back to step (1).

(3) If the BHT receives any data, it will check the data. If the BHT detects a “WAKE” character string[*1] in the data, it will proceed; if not, the BHT will go back to step (1).

(4) The BHT will send the following response to the host computer depending upon whether or not a remote wakeup program exists in the BHT and whether the remote wakeup function is activated or deactivated.

Remote wakeup program	Remote wakeup function	Response message from the BHT	Proceeds to:
Exists	Activated	ACK + "0" + ID[*2]	(5)
	Deactivated	ACK + "2" + ID	(1)
Not exist	Activated	ACK + "1" + ID	(1)
	Deactivated	ACK + "1" + ID	(1)

[*1] Since the BHT in the charge state is in sleep, it will not receive the 1st “WAKE” character string normally. To wake up the BHT, you need to send a “WAKE” character string at lease two times, for example, send “WAKEWAKE” or “WAKEWAKEWAKE.”

[*2] ID is a 6-byte numeric string referring to the lower 6 digits of the BHT serial number.

(5) The BHT will exit from the sleep state and execute the remote wakeup program developed by the user.

■ At the host computer

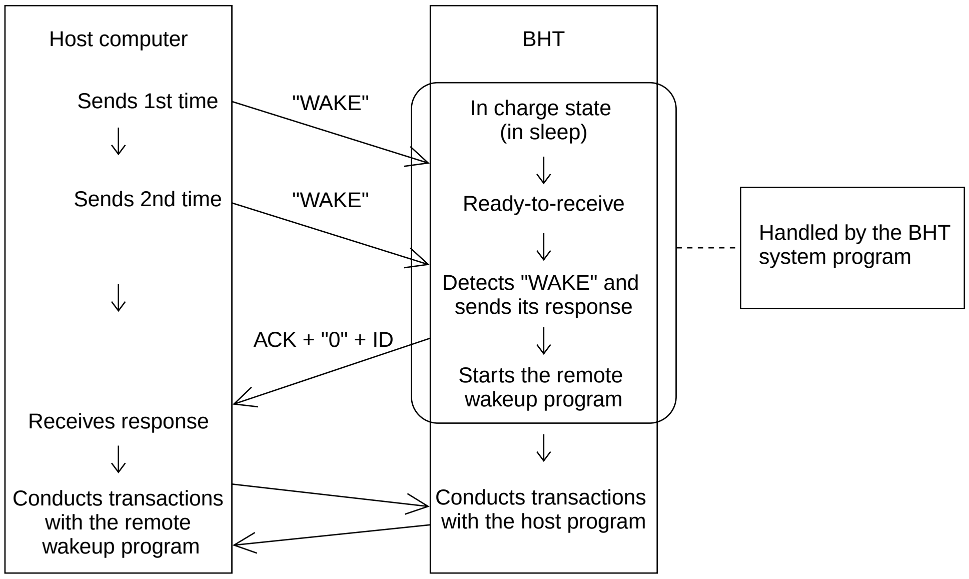
(1) The host computer sends a “WAKE” character string to the BHT at least two times.

(2) Upon receipt of "ACK + "0" + ID" from the BHT, the host computer should conduct transactions with the remote wakeup program in the BHT.

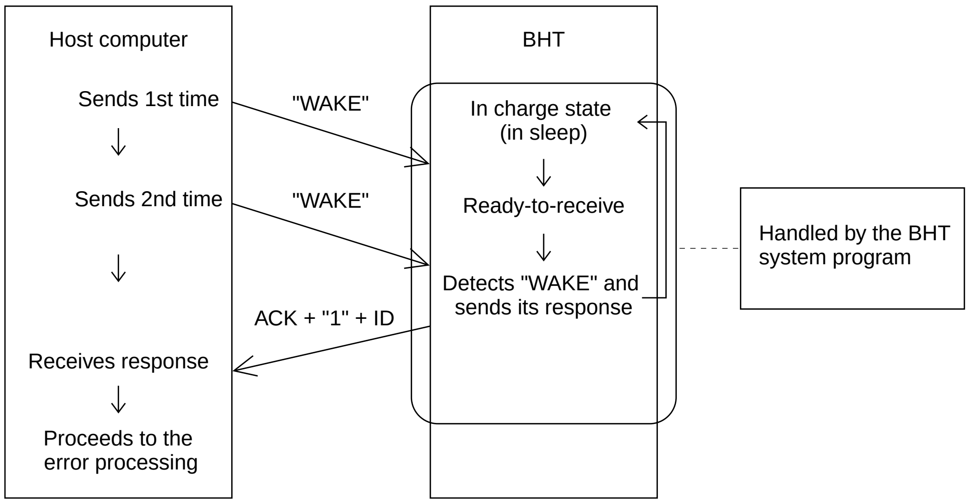
Upon receipt of "ACK + "1" + ID" or "ACK + "2" + ID" from the BHT, the host computer should proceed to the corresponding error processing.

■ Transmission control sequence

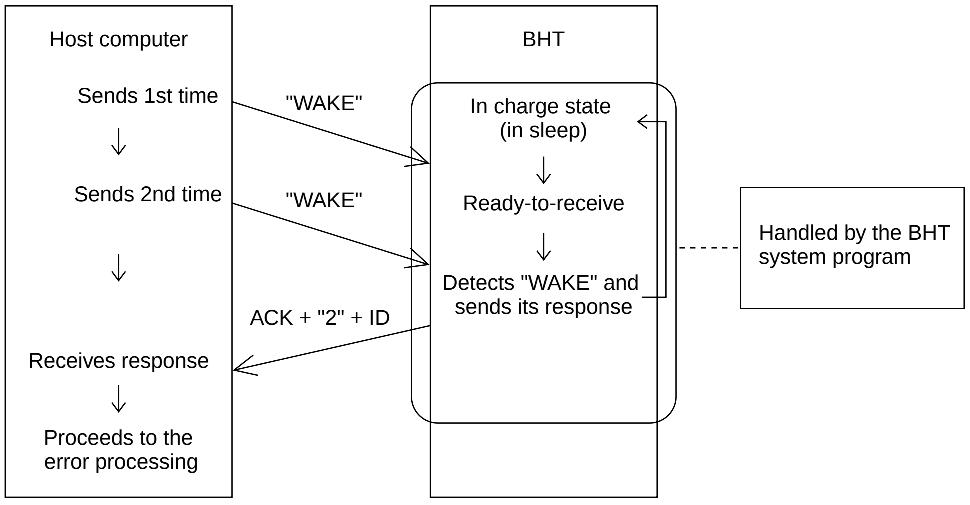
If in the BHT a remote wakeup program exists and the remote wakeup is activated:


Host computer
BHT
Sends 1st time
"WAKE"
In charge state (in sleep)
Sends 2nd time
"WAKE"
Ready-to-receive
Handled by the BHT system program
Detects "WAKE" and sends its response
ACK + "0" + ID
Starts the remote wakeup program
Receives response
Conducts transactions with the remote wakeup program
Conducts transactions with the host program

If in the BHT no remote wakeup program exists:


Host computer
Sends 1st time
Sends 2nd time
Receives response
Proceeds to the error processing
"WAKE"
"WAKE"
ACK + "1" + ID
BHT
In charge state (in sleep)
Ready-to-receive
Detects "WAKE" and sends its response
Handled by the BHT system program

If in the BHT the remote wakeup is deactivated:


Host computer
Sends 1st time
Sends 2nd time
Receives response
Proceeds to the error processing
"WAKE"
"WAKE"
ACK + "2" + ID
BHT
In charge state (in sleep)
Ready-to-receive
Detects "WAKE" and sends its response
Handled by the BHT system program

[3] Remote wakeup program

■ File name

The BHT may handle the file named "BHTRMT.PD3" as a remote wakeup program.

Upon receipt of data containing a “WAKE” character string in the ready-to-receive state, the BHT checks whether the BHTRMT.PD3 file exists. If the file exists, the BHT will start the remote wakeup operation described in [2].

■ Settings for remote wakeup

To use the remote wakeup function, make the following I/O port settings with the `OUT` or `WAIT` statement or `INP` function beforehand (refer to Appendix D, "I/O Ports," D5):

(1) Activate the remote wakeup function

You may activate/deactivate the remote wakeup function as listed below. The default is 0 (Deactivate).

Port No.	Bit No.	R/W	Specifications
60F0h	0	R/W	0: Deactivate the remote wakeup 1: Activate the remote wakeup

(2) Set the transmission speed to be applied for remote wakeup

Set the transmission speed to be applied when activating the remote wakeup as listed below. The default is 1 (9600 bps).

Port No.	Bit No.	R/W	Specifications
60F1h	7-0	R/W	1: 9600 bps 2: 19200 bps 3: 38400 bps 4: 57600 bps 5: 115200 bps

(3) Set the timeout for ready-to-receive state

Set the timeout length during which the BHT will wait for a “WAKE” character string after receiving any data via the CU and becoming ready to receive. The default is 3 (seconds).

Port No.	Bit No.	R/W	Specifications
60F3h	7-0	R/W	1 to 255 seconds. Specification of 0 will not change the current setting.

(4) Set the BHT station ID to be used in the BHT response message

Set a 6-byte numeric string referring to the lower 6 digits of the BHT serial number as a station ID which will be used in the response message to the host. To write and read the setting, use the extension function SYSTEM.FN3 (Functions #3 and #4). For details, refer to Chapter 16, "Extended Functions."

Once made in a user program, the above settings will be retained even after termination the user program.

The remote wakeup activation/deactivation and the transmission speed for remote wakeup may be set in System Mode. For details, refer to the "BHT-7000 User’s Manual" or "BHT-7500 User’s Manual."

■ Start of a remote wakeup program

When a remote wakeup program starts, the resume function of the most recently running user program becomes disabled regardless of the resume setting made in System Mode. Also in other user programs chained from the remote wakeup program with the `CHAIN` statement, the resume function will remain disabled.

Accordingly, after termination of the remote wakeup program, any other user program will perform a cold start.

To enable the resume function of a user program running after the termination of the remote wakeup program and its chained-to programs, use the extension function SYSTEM.FN3 (Function #1). For details, refer to Chapter 16, "Extension Functions."

■ End of a remote wakeup program

The remote wakeup program and its chained-to programs may be either normally terminated or interrupted as follows:

- Normally terminated

  when the program is ended with `END`, `POWER OFF` or `POWER 0` statement.

- Interrupted

  when the program is ended by pressing the PW key, with automatic powering-off function, low battery power-off or any other factor when the resume function is disabled.

If the resume function is made enabled, the remote wakeup program or its chained-to program will be neither normally terminated nor interrupted since it will resume the operation in the next powering-on.

■ Checking the execution record of remote wakeup

When starting, a user program (including a remote wakeup program) may check via the I/O ports whether the BHT remotely woke up at the last powering on and its operation was normally ended. (Refer to Appendix D, "I/O Ports," D5.)

Making use of the execution record, you may display an alarm message.

Port No.	Bit 0	Bit 1	Specifications
60F2h	0	0	At the last powering on, the BHT did not remotely wake up.*
	0	1	
	1	0	At the last powering on, the BHT remotely woke up and its operation was interrupted.
	1	1	At the last powering on, the BHT remotely woke up and its operation was normally ended.

* This means that the BHT was cold-started, driven by System Mode or initialized.

Chapter 13
LCD Backlight Function

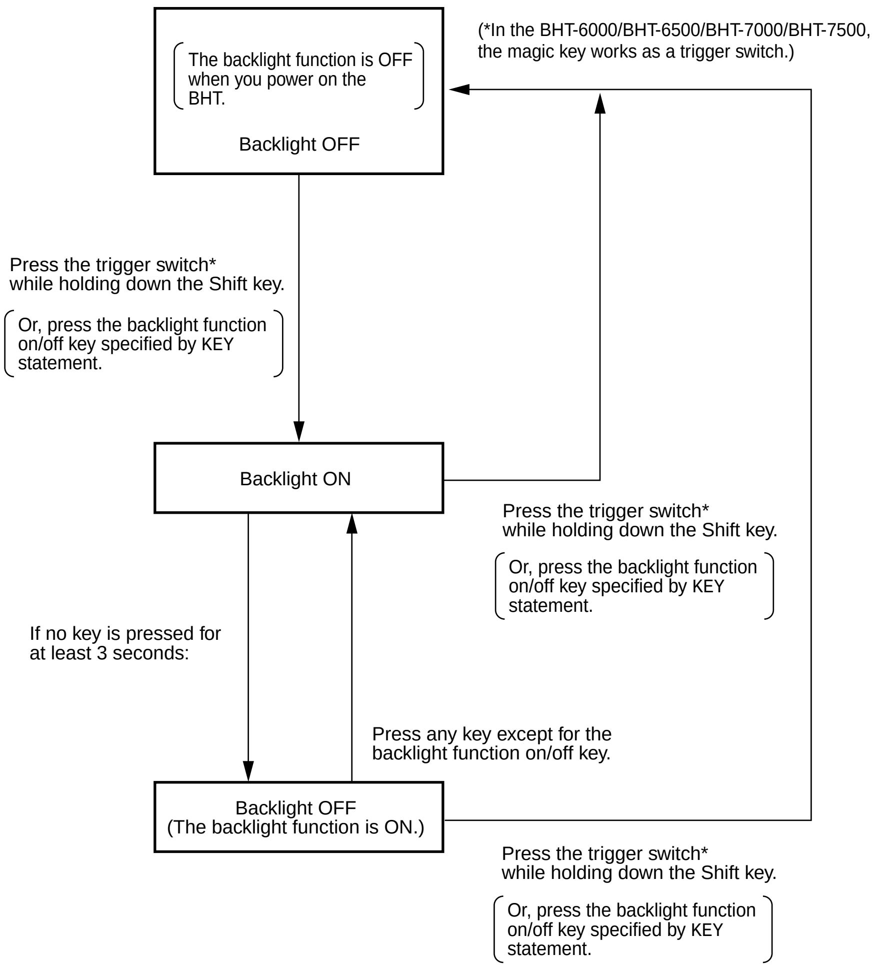
CONTENTS

13.1 LCD Backlight Function

The BHT has an LCD backlight function. Pressing the trigger switch* while holding down the Shift key activates or deactivates the backlight function. The default length of backlight ON-time (ON-duration) is 3 seconds.

By using a KEY statement, you can select the backlight function on/off key instead of the combination of the trigger switch* and Shift key, as well as modifying the ON-duration of the backlight.

For details about the KEY statement, refer to KEY in Chapter 14.

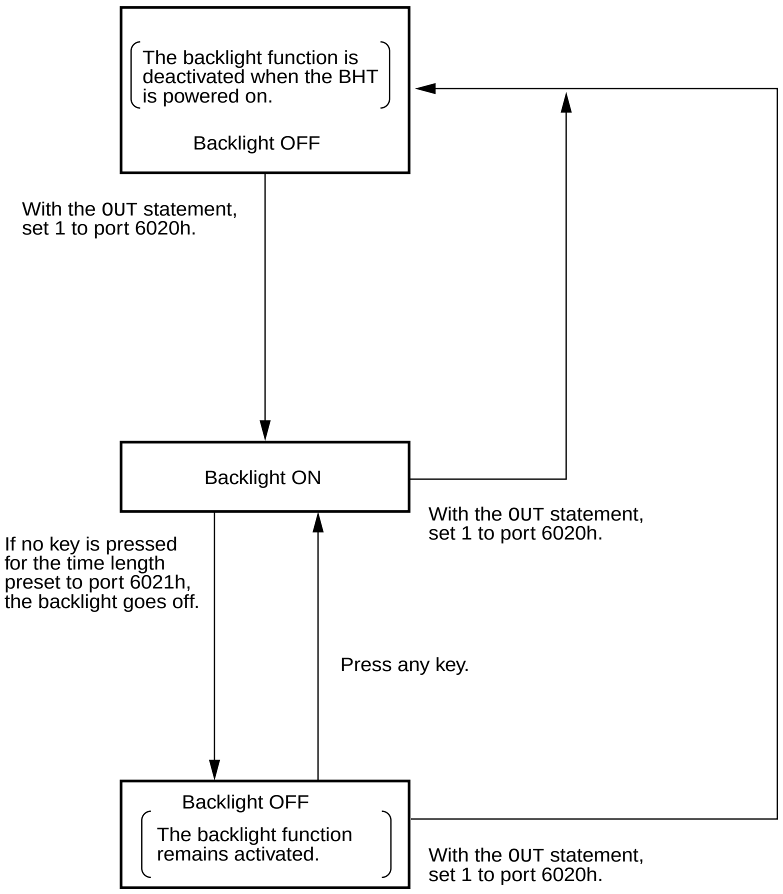

(*In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, the magic key works as a trigger switch.)
The backlight function is OFF when you power on the BHT.
Backlight OFF
Press the trigger switch* while holding down the Shift key.
Or, press the backlight function on/off key specified by KEY statement.
Backlight ON
Press the trigger switch* while holding down the Shift key.
Or, press the backlight function on/off key specified by KEY statement.
If no key is pressed for at least 3 seconds:
Press any key except for the backlight function on/off key.
Backlight OFF (The backlight function is ON.)
Press the trigger switch* while holding down the Shift key.
Or, press the backlight function on/off key specified by KEY statement.

In the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, you can control the backlight function by using the OUT statement as described below.

Setting 1 to port 6020h with the OUT statement activates the LCD backlight function and turns on the backlight. If no key is pressed for the time length preset to port 6021h (default time: 5 seconds), the backlight goes off but the backlight function remains activated.

Setting 0 to port 6020h deactivates the LCD backlight function and turns off the backlight if lit.

When the backlight function is activated with the OUT statement, the backlight function on/off key and ON-duration specified by the KEY statement will be ignored.


The backlight function is deactivated when the BHT is powered on.
Backlight OFF
With the OUT statement, set 1 to port 6020h.
Backlight ON
With the OUT statement, set 1 to port 6020h.
If no key is pressed for the time length preset to port 6021h, the backlight goes off.
Press any key.
Backlight OFF
The backlight function remains activated.
With the OUT statement, set 1 to port 6020h.

Chapter 14
Statement Reference

CONTENTS

ANK Pattern LOAD I/O statement

APLOAD

Loads a user-defined font in the single-byte ANK* mode

*ANK: Alphanumeric and Katakana

Syntax:

Syntax 1 (Loading a user-defined font):

`APLOAD` *`characode,fontarrayname`*

Syntax 2 (Loading a user-defined cursor. Valid in the BHT-7000/BHT-7500):

`APLOAD` *`characode,cursorarrayname`*

Parameter:

`characode`

• For user-defined font	A numeric expression which returns a value from 128 (80h) to 159 (9Fh).
• For user-defined cursor	A numeric expression which returns a value 0.

`fontarrayname` and *`cursorarrayname`*

An array integer variable name.

NOTE Do not specify parentheses ( ) or subscripts which represent a general array as shown below; otherwise, it will result in a syntax error.

```
APLOAD &H80,cp%()    'error
APLOAD &H80,cp%(5)   'error
```

Description:

■ Loading a user-defined font

`APLOAD` loads a user-defined font data defined by *`fontarrayname`* to the user font area specified by *`characode`*.

- To display user-defined fonts loaded by the `APLOAD`, you use the `PRINT` statement in the single-byte ANK mode. If you attempt to display an undefined character code, a space character will appear.
- The loaded user-defined fonts are effective during execution of the user program which loaded those fonts and during execution of the successive user programs chained by the `CHAIN` statement.

- If you issue more than one `APLOAD` statement specifying a same character code, the last statement takes effect.
- Only when the Interpreter executes the `APLOAD` statement, it refers to the array data defined by *`fontarrayname`*. So, once a user program has finished loading the user font, changing the data in the array or deleting the array itself (by the `ERASE` statement) will not affect the already loaded user font.
- An array integer variable--a work array, register array, or common array--for *`fontarrayname`* should be declared by the `DIM`, `DEFREG`, or `COMMON` statement, respectively.

```
DIM cp0%(5)
DEFREG cp1%(5)
COMMON cp2%(5)
```

  The array variable should be one-dimensional and have at least six elements. Each element data should be an integer and stored in the area from the 1st to 6th elements of the array.

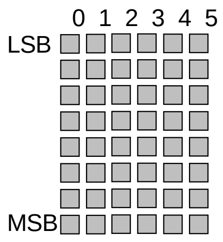
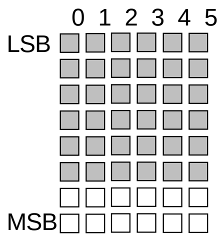
- In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, if the small-size font is selected, user-defined fonts loaded by the `APLOAD` will be condensed into small size (6 dots wide by 6 dots high) for display. For the generating procedure of the small-sized user-defined fonts, refer to Appendix C3., "Display Mode and Letter Size."
- Also in the double-width screen mode of the BHT-7000/BHT-7500, user-defined fonts loaded by the `APLOAD` will be effective, but the dot pattern of each loaded font will be doubled in width by the system.

■ Loading a user-defined cursor (BHT-7000/BHT-7500)

`APLOAD` loads a user-defined cursor data defined by *`cursorarrayname`* to the user font area specified by *`characode`*.

- To display a user-defined cursor loaded by the `APLOAD`, you specify 255 to the *`cursorswitch`* in the `LOCATE` statement in the single-byte ANK mode. (`LOCATE ,,255`)
- The loaded user-defined cursors are effective during execution of the user program which loaded those cursors and during execution of the successive user programs chained by the `CHAIN` statement.
- Only when the Interpreter executes the `APLOAD` statement, it refers to the array data defined by *`cursorarrayname`*. So, once a user program has finished loading the user cursor, changing the data in the array or deleting the array itself (by the `ERASE` statement) will not affect the already loaded user cursor.

- The cursor size will differ depending upon the display font currently selected, as shown below.

Display font	Size (W x H)	No. of elements
Standard-size	6 x 8 dots 	6
Small-size	6 x 6 dots 	6

- An array integer variable--a work array, register array, or common array--for *`cursorarrayname`* should be declared by the `DIM`, `DEFREG`, or `COMMON` statement, respectively.

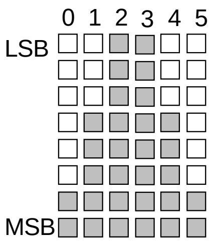
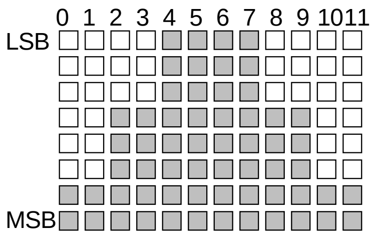
```
DIM cp0%(5)
DEFREG cp1%(5)
COMMON cp2%(5)
```

  The array variable should be one-dimensional and have at least six elements. Each element data should be an integer and stored in the area from the 1st to 6th elements of the array.

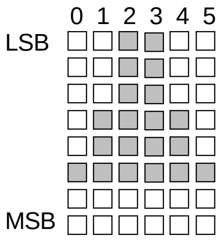
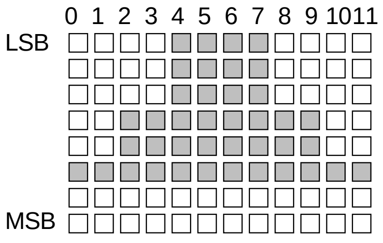
- If you specify *`cursorarrayname`* exceeding the allowable cursor size (height: no. of bits, width: no. of elements), the excess will be discarded.

- In the double-width screen mode, user-defined cursors loaded by the `APLOAD` will be doubled in width when displayed, as shown below.

  When the standard-size font is selected:

Cursor loaded	In double-width screen mode
	

  When the small-size font is selected:

Cursor loaded	In double-width screen mode
	

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• No *`fontarrayname`* or *`cursor-arrayname`* is defined. • *`fontarrayname`* or *`cursorar-rayname`* has an array string vari-able. • *`fontarrayname`* or *`cursorar-rayname`* includes parentheses ( ). • *`fontarrayname`* or *`cursorar-rayname`* includes subscripts.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (• *`characode`* is out of the specified range.) (• The array structure is not correct.)
`08h`	Array not defined

Example:

```
DIM cp%(5)
cp%(0)=&H00
cp%(1)=&H08
cp%(2)=&H1C
cp%(3)=&H3E
cp%(4)=&H7F
cp%(5)=&H00
APLOAD &H80,cp%
PRINT CHR$(&H80)
```

Array Elements						
cp%(0)	cp%(1)	cp%(2)	cp%(3)	cp%(4)	cp%(5)	Bit in each array element
□	□	□	□	■	□	0(LSB)
□	□	□	■	■	□	1
□	□	■	■	■	□	2
□	■	■	■	■	□	3
□	□	■	■	■	□	4
□	□	□	■	■	□	5
□	□	□	□	■	□	6
□	□	□	□	□	□	7(MSB)

Reference:

Statements: `COMMON`, `DEFREG`, `DIM`, `KPLOAD`, `PRINT`, and `SCREEN`

BEEP

Drives the beeper or vibrator. (The vibrator is provided in the BHT-6500/BHT-7000/BHT-7500 to which vibrator-related descriptions given below should apply.)

Syntax:

```
BEEP[onduration[,offduration[,repetitioncount
[,frequency]]]]
```

Parameter:

onduration, *offduration*, and *repetitioncount*

Numeric expressions, each of which returns a value from 0 to 255.

frequency

A numeric expression which returns a value from 0 to 32767.

Description:

BEEP sounds the beeper or drives the vibrator during the length of time specified by *onduration* at the intervals of the length of time specified by *offduration* by the number of repetitions specified by *repetitioncount*.

The beeper sounds at the pitch of the sound in Hz specified by *frequency*.

- The unit of *onduration* and *offduration* is 100 msec.
- Defaults:

onduration and *offduration*:	1 (100 msec.)
repetitioncount:	1
frequency:	4337 Hz* (BHT-3000/BHT-6000) 3213 Hz* (BHT-4000) 4200 Hz* (BHT-5000) 2711 Hz* (BHT-6500) 2793 Hz* (BHT-7000/BHT-7500) (*Same as when 2 is set to *frequency*)

- Note that specification of 0, 1, or 2 to *frequency* produces the special beeper effects as listed below.

Specification to *frequency*	BHT-3000/ BHT-6000	BHT-4000	BHT-5000	BHT-6500	BHT-7000/ BHT-7500	Tone	Statement example
0	1033 Hz	1015 Hz	1015 Hz	986 Hz	698 Hz	Low-pitched	`BEEP ,,,0`
1	2168 Hz	1752 Hz	2142 Hz	1807 Hz	1396 Hz	Medium-pitched	`BEEP ,,,1`
2	4337 Hz	3213 Hz	4200 Hz	2711 Hz	2793 Hz	High-pitched	`BEEP ,,,2`

In the BHT-6500/BHT-7000/BHT-7500, specification of 0, 1, or 2 to *frequency* drives the beeper or vibrator depending upon the settings made on the "LCD contrast & beeper volume adjustment and the beeper & vibrator switching" screen.

If 0, 1, or 2 is set to *frequency* (or if the *frequency* option is omitted), then you can adjust the beeper volume on the LCD when powering on the BHT. (For the adjustment procedure, refer to the BHT User's Manual.)

In the BHT-7000/BHT-7500, you may change the beeper volume with the `OUT` statement. (For details, refer to Appendix D, "I/O Ports," D5.)

If you set a value other than 0, 1, and 2 to *frequency*, the beeper volume is automatically set to the maximum and not adjustable.

- In the BHT-3000/BHT-6000/BHT-6500, specification of any of 3 through 61 to *frequency* deactivates the beeper; in the BHT-4000/BHT-5000, any of 3 through 260 deactivates the beeper; in the BHT-7000/BHT-7500, any of 3 through 39 deactivates the beeper or vibrator.
- In the BHT-4000, specification of 5001 or greater to *frequency* automatically sets the frequency to 5000 Hz.
- Specification of zero to *onduration* deactivates the beeper.
- Specification of a value except for zero to *onduration* and specification of zero to *offduration* keep beeping.
- Specification of a value except for zero to *onduration* and *offduration* and specification of zero to *repetitioncount* deactivate the beeper.
- For your reference, the relationship between the frequencies and the musical scale is listed below.

	Scale 1	Scale 2	Scale 3	Scale 4	Scale 5	Scale 6
do	130 Hz	261 Hz	523 Hz	1046 Hz	2093 Hz	4186 Hz
do#	138	277	554	1108	2217	
re	146	293	587	1174	2349	
re#	155	311	622	1244	2489	
mi	164	329	659	1318	2637	
fa	174	349	698	1396	2793	
fa#	184	369	739	1479	2959	
sol	195	391	783	1567	3135	
sol#	207	415	830	1661	3322	
la	220	440	880	1760	3520	
la#	233	466	932	1864	3729	
si	246	493	987	1975	3951	

- The BEEP statement does not suspend execution of the subsequent statement until the beeper completes sounding or vibrating. Instead, the execution of the subsequent statement proceeds immediately.

  If a second BEEP statement is encountered while the BHT is still beeping or vibrating by a first BEEP, the first BEEP is cancelled and the new BEEP statement executes.
- In the BHT-3000, if the beeper starts sounding for warning you of the low battery during beeping programmed by the BEEP, then the warning beep overrides the programmed beeping.
- In the BHT-6000/BHT-6500/BHT-7000/BHT-7500 also, if low battery warning operation starts during beeping or vibrating programmed by the BEEP, then the warning operation overrides the programmed beeping or vibrating. Upon completion of the warning operation, the beeper or vibrator resumes working as programmed.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	The number of parameters or commas (,) exceeds the limit.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range

Example:

```
BEEP bon%,boff%,count%,helz%
BEEP bon%,boff%,count%
BEEP bon%,boff%,,helz%
BEEP bon%,,count%,helz%
BEEP ,boff%,count%,helz%
BEEP bon%,boff%
BEEP bon%,,count%
BEEP ,boff%,count%
BEEP bon%,,,helz%
BEEP ,boff%,,helz%
BEEP ,,count%,helz%
BEEP bon%
BEEP ,boff%
BEEP ,,count%
BEEP ,,,helz%
BEEP
```

Flow control statement

CALL

Calls an FN3 or SUB function.

Syntax:

Syntax 1 (Calling an FN3):

CALL "[*drivename*:]*filename*" *functionnumber* [*data* [,*data*]...]

Syntax 2 (Calling a SUB):

CALL *functionname* [(*realparameter*[,*realparameter*…])]

Parameter:

[*drivename*:]*filename*

A string expression.

functionnumber

An integer constant.

data

A string variable or a numeric variable.

functionname

Real function name.

realparameter

A numeric expression or a string expression.

Description:

■ Calling an extension library (FN3 function)

CALL calls a function specified by *functionnumber* from a file specified by "[*drivename*:]*filename*" and assigns the parameter specified by *data* to the called function.

For the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, the *drivename* may be A: or B:. If the *drivename* is omitted, the default A: applies.

- `filename` is the name of an FN3 function. The extension of the file names is fixed to .FN3. (For the FN3 functions, refer to Chapter 16, "Extended Functions" or the "BHT-BASIC Extension Library Manual.")
- `functionnumber` is the function number of an FN3 specified by "`[drive-name:]filename`".
- `data` is a variable for the function number of the FN3 (that is, it is used as an argument to the FN3 function).
- When specifying an array to `data`, add a pair of parentheses containing nothing as shown below.

  Example: `CALL "_xxx.FN3" 1 DATA ()`
- When calling a function (specified by `functionnumber`) that returns a string variable:

  BHT-5000/BHT-6000/BHT-6500

  Reserve a storage area for a returned string variable by using a variable declaration statement (`DIM`, `COMMON`, or `DEFREG`). (If the string length is omitted in the variable declaration statement, the default in the statement will take effect.) And assign arbitrary data of the string length required for a return value to the variable.

  If the string length of a returned value is greater than the assigned string length, a run-time error will result.

  (Example 1) If a return value is a fixed-length string, e.g. 8-character length:

  ```
  DIM OUTPUT$[8]            'Reserves a storage area of 8 characters.
  OUTPUT$="        "        'Assigns 8 spaces.
  ```

  (Example 2) If a return value is a variable-length string of a maximum of N characters:

  ```
  DIM OUTPUT$[N]            'Reserves a storage area of a max. of N chars.
  OUTPUT$=" ... "           'Assigns N spaces.
  OUTPUT$=""                'or assign
  FOR I%=1 TO N             'a max. of N chars of spaces
      OUTPUT$=OUTPUT$+""    'to the variable by loop.
  NEXT I%
  ```

  BHT-7000/BHT-7500

  Reserve a storage area for a returned string variable by using a variable declaration statement (`DIM`, `COMMON`, or `DEFREG`). (If the string length is omitted in the variable declaration statement, the default in the statement will take effect.) Unlike the BHT-5000/BHT-6000/BHT-6500, the BHT-7000/BHT-7500 does not require to assign arbitrary data of the string length required for a return value to the variable.

  If the string length of a returned value is greater than the string length reserved by a variable declaration statement, a run-time error will result.

  (Example 1) If a return value is a fixed-length string, e.g. 8-character length:

  ```
  DIM OUTPUT$[8]            'Reserves a storage area of 8 characters.
  ```

  (Example 2) If a return value is a variable-length string of a maximum of N characters:

  ```
  DIM OUTPUT$[N]            'Reserves a storage area of a max. of N chars.
  ```

NOTE To use FN3 functions except extended functions given in Chapter 16, you need to download the extension programs from an extension library sold separately. (The BHT-BASIC Extension Library is supported by the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500.)

■ Calling a user-defined function (SUB function)

This statement calls a user-defined function specified by *functionname*. You may omit CALL when calling a SUB function.

- *functionname* should be a user-defined function defined by SUB...END SUB statement.
- The number of *realparameter*s should be equal to that of dummy parameters, and the types of the corresponding variables used in those parameters should be identical.
- If you specify a global variable in *realparameter* when calling a user-defined function, the user-defined function cannot update the value of the global variable. This is because all *realparameter*s are passed not by address but by value. (So called "Call-by-value")

NOTE Before any call to a SUB function, you need to place definition of the SUB function or declaration of the SUB function by using the DECLARE statement in your source program.

Syntax errors:

Error code and message	Meaning
`error 3:  '"' missing`	No double quote precedes or follows [*drivename*:]*filename*.
`error 68: Mismatch`	• The number of *realparameter*s is not equal to that of the dummy parameters. • A dummy parameter was an integer variable in defining a function, but *realparameter* is a real type in calling the function. (If a dummy parameter was a real variable in defining a function and *realparameter* is an integer type in calling, then no error occurs.)
`error 71: Syntax error`	• [*drivename*:]*filename* is not enclosed in double quotes. • The function specified by *functionname* has not been defined.

Run-time errors:

Error code	Meaning
`02h`	Syntax error ("`[`*`drivename`*`:]`*`filename`*" is in incorrect syntax or the extension is not .FN3.)
`05h`	Parameter value out of range (In calling an FN3 function, the number of parameters exceeds 16.)
`07h`	Insufficient memory space (You nested calling statements of a user-defined function to more than 10 levels.)
`1Fh`	*`functionnumber`* out of the range
`35h`	File not found
`F0h`	Mismatch parameter number
`F1h`	Mismatch parameter type
`F2h`	Insufficient string variable storage area

Reference:

Statements: `DECLARE` and `SUB`...`END SUB`

Flow control statement

CHAIN

Transfers control to another program.

Syntax:

`CHAIN "[`*`drivename`*`:]`*`programfilename`*`"`

Parameter:

`"[`*`drivename`*`:]`*`programfilename`*`"`

A string expression.

Description:

`CHAIN` transfers control to a program specified by `"[`*`drivename`*`:]`*`program-filename`*`"`. That is, it terminates the current running program (1st program) and closes all of the files being opened. Then, it initializes environments for the chained-to user program (2nd program) specified by `"[`*`drivename`*`:]`*`programfile-name`*`"` and executes it.

For the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, the *`drivename`* may be `A:` or `B:`. If the *`drivename`* is omitted, the default `A:` applies.

- `"[`*`drivename`*`:]`*`programfilename`*`"` is an executable object program compiled by the Compiler and has the extension .PD3, as shown below. The extension .PD3 cannot be omitted.

```
CHAIN "prog1.PD3"
```

- You should download an executable object program (2nd program) to the BHT before the `CHAIN` statement is executed.
- You can pass variables from the current program to the chained-to program (2nd program) with the `COMMON` statement.
- User-defined fonts loaded by the `APLOAD` or `KPLOAD` statement and the setting values assigned by the `KEY` statement or `COUNTRY$` function remain effective in chained-to programs.
- The `ON ERROR GOTO` statement cannot trap errors (while showing the error code 07h which means "Insufficient memory space") happened during initialization of environments for chained-to programs.

Syntax errors:

Error code and message	Meaning
`error 3: ’"’ missing`	No double quote precedes or follows `[`*`drivename`*`:]`*`programfile-name`*.
`error 71: Syntax error`	`[`*`drivename`*`:]`*`programfile-name`* is not enclosed in double quotes.

Run-time errors:

Error code	Meaning
`02h`	Syntax error ("`[`*`drivename`*`:]`*`programfilename`*" is in incorrect sytax or the extension is not .PD3.)
`07h`	Insufficient memory space (The 1st program uses too many variables.)
`35h`	File not found (The file specified by "`[`*`drivename`*`:]`*`programfilename`*" does not exist.)
`41h`	File damaged

Reference:

Statements: `APLOAD`, `COMMON`, and `KPLOAD`

CLear FILE File I/O statement

CLFILE

Erases the data stored in a data file.

Syntax:

```
CLFILE [#]filenumber
```

Parameter:

filenumber

A numeric expression which returns a value from 1 to 16.

Description:

`CLFILE` erases data in the data file specified by *filenumber* and resets the number of written records in the directory to zero.

- The memory area freed by `CLFILE` can be used for other data files or user program files.
- User programs can no longer refer to the erased data.
- `CLFILE` cannot erase data in files stored in drive B.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	*filenumber* is missing.

Run-time errors:

Error code	Meaning
`34h`	Bad file name or number (You specified *filenumber* of an unopened file.)
`36h`	Improper file type (You specified *filenumber* of a file other than data files.)
`3Ah`	File number out of the range
`43h`	Not allowed to access the data in drive B.

Example:

```
OPEN "master.Dat" AS #1
FIELD #1,20 AS bar$,10 AS ky$
CLFILE #1
CLOSE #1
```

'File I/O statement

CLOSE

Closes file(s).

Syntax:

```
CLOSE [[#]filenumber[,[#]filenumber...]]
```

Parameter:

filenumber
: A numeric expression which returns a value from 1 to 16.

Description:

`CLOSE` closes file(s) specified by *filenumber*(s).

- The file number(s) closed by the `CLOSE` statement becomes available for a subsequent `OPEN` statement.
- If no file number is specified, the `CLOSE` statement closes all of the opened data files and device I/O files.
- Specifying the unopened file number causes neither operation nor a run-time error.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	*filenumber* is missing.

Run-time errors:

Error code	Meaning
3Ah	File number out of range

Reference:

Statements:	`END` and `OPEN`

CLS

Clears the LCD screen.

Syntax:

```
CLS
```

Description:

`CLS` clears the liquid crystal display (LCD) screen and returns the cursor to the upper left corner of the screen.

- The `CLS` statement does not affect the screen mode (the single-byte ANK mode, two-byte Kanji mode, and condensed two-byte Kanji mode) or the character attribute (normal or highlighted), but it turns off the cursor.
- In the BHT-4000/BHT-5000/BHT-6000/BHT-6500, execution of the `CLS` statement when the system status is displayed on the LCD clears the VRAM area assigned to the system status area of the LCD but does not erase the system status displayed.

Declarative statement

COMMON

Declares common variables for sharing between user programs.

Syntax:

```
COMMON commonvariable[,commonvariable...]
```

Parameter:

commonvariable

A non-array integer variable, a non-array real variable, a non-array string variable, an array integer variable, an array real variable, or an array string variable.

Description:

`COMMON` defines common variables for sharing them when one program chains to another.

- Common variables defined by `COMMON` keep effective as long as programs chained by the `CHAIN` statement are running.
- A `COMMON` statement can appear anywhere in a source program.
- All of the variable name, type, quantity, and definition order of the common variables used in the current program should be identical with those in the chained-to programs. If not, variables having indefinite values will be passed.
- Up to two-dimensional array variables can be defined. You can specify a subscript ranging from 0 to 254 for an array variable.
- The total variable data size which can be passed between chained programs is 6 kilobytes including work variables.
- The size of an array data is equal to the element size multiplied by the number of elements.
- You can specify the maximum string length within the range from 1 to 255 to a string variable.
- The default length of a non-array string variable is 40.
- The default length of an array string variable is 20.

Syntax errors:

Error code and message	Meaning
`error 5: Variable name redefinition`	A same variable name is double declared in a program.
`error 73: Improper string length`	The length of a string variable is out of the range from 1 to 255.

Run-time errors:

Error code	Meaning
`07h`	Insufficient memory space (The `COMMON` statement defines too much data.)

Example:

```
COMMON a%,b,c$,d%(2,3),e(4),f$(5)
```

Reference:

Statements:	`CHAIN`

Declarative statement

CONST

Defines symbolic constants to be replaced with labels.

Syntax:

```
CONST constname = expr
```

Parameter:

constname
: A label, identifier, or string expression of a maximum of 10 characters consisting of alphanumerics and period (.).

expr
: A string constant

Description:

`CONST` replaces a label, identifier or a character string specified by *constname* with a string constant defined by *expr* before compiling.

- *expr* may contain labels defined by other `CONST` declarations. However, calling those labels each other (recursively) will result in an error.
- A `CONST` statement can appear anywhere in your source program. However, it will take effect from a program line following the `CONST` declaration.

CURSOR

Turns the cursor on or off.

Syntax:

```
CURSOR {ON|OFF}
```

Description:

When a user program is initiated, the cursor is set to OFF. `CURSOR ON` turns on the cursor for keyboard entry operation by the `INKEY$` function. `CURSOR OFF` turns off the cursor.

- In the BHT-3000/BHT-4000/BHT-5000, the cursor size depends upon the screen mode (the single-byte ANK mode, two-byte Kanji mode, or condensed two-byte Kanji mode). In the single-byte ANK mode, the cursor appears as 6 dots wide by 8 dots high; in the two-byte Kanji mode, it appears as 8 dots wide by 16 dots high; in the condensed two-byte Kanji mode, it appears as 6 dots wide by 16 dots high. (Note that the condensed two-byte Kanji mode is supported by the BHT-4000/ BHT-5000.)
- In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, the cursor size depends upon the screen mode (the single-byte ANK mode or two-byte Kanji mode), the display font size (standard-size or small-size), and the character attribute (normal-width or double-width). If the standard-size font is selected, the cursor appears as 6 dots wide by 8 dots high in the single-byte ANK mode, and as 8 dots wide by 16 dots high in the two-byte Kanji mode. If the small-size font is selected, the cursor appears as 6 dots wide by 6 dots high in the single-byte ANK mode, and as 6 dots wide by 12 dots high in the two-byte Kanji mode.
- The cursor shape specified by the most recently executed `LOCATE` statement takes effect.
- After execution of `LOCATE ,,0` which makes the cursor invisible, even the `CURSOR ON` statement cannot display the cursor. To display the cursor, it is necessary to make the cursor visible by using the `LOCATE` statement.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	Specification other than `ON` and `OFF` is described.

Reference:

Statements:	`APLOAD`, `INPUT`, `KPLOAD`, `LINE INPUT`, and `LOCATE`
Functions:	`INKEY$` and `INPUT$`

Declarative statement

DATA

Stores numeric and string literals for READ statements.

Syntax:

```
DATA literal[,literal...]
```

Parameter:

literal

A numeric or string constant.

Description:

DATA stores numeric and string literals so that READ statements can assign them to variables.

- A DATA statement can appear anywhere in a source program.
- A string data should be enclosed with a pair of double quotation marks (").
- You may have any number of DATA statements in a program. The READ statement assigns data stored by DATA statements in the exact same order that those DATA statements appear in a source program.
- Using the RESTORE statement can read a same DATA statement more than once.
- You can specify more than one *literal* in a program line (within 512 characters) by separating them with commas (,).
- You can describe DATA statements also in included files.

Syntax errors:

Error code and message	Meaning
`error 3:’"’ missing`	No double quote precedes or follows a string data.

Reference:

Statements: READ, REM and RESTORE

DECLARE

Declares user-created function FUNCTION or SUB externally defined.

Syntax:

Syntax 1 (Defining a numeric FUNCTION):

DECLARE FUNCTION *funcname*
[(*dummyparameter*[,*dummyparameter*...])]

Syntax 2 (Defining a character FUNCTION):

DECLARE FUNCTION *funcname* [(*dummyparameter*
[,*dummyparameter*...])][[*stringlength*]]

Syntax 3 (Defining a SUB):

DECLARE SUB *subname*[(*dummyparameter*
[,*dummyparameter*...])]

Parameter:

funcname

- For numerics

funcname%	Integer function name
funcname	Real function name

- For strings

funcname$	Character function name

subname

Real function name.

dummyparameter

A non-array integer variable, a non-array real variable, or a non-array string variable.

stringlength

An integer constant having a value from 1 to 255.

Description:

`DECLARE` defines a user-created function defined in other source program files.

- Declaration of a user-defined function should appear preceding a calling statement of the user-defined function in your source program.
- *`funcname`*, *`subname`*, and *`dummyparameter`* should be declared in the same way as the function names and real parameters defined in the original functions (defined in other source program files).
- You cannot make double definition to a same function name.
- The `DECLARE` statement should not be defined in the block-structured statements (`FOR`...`NEXT`, `IF`...`THEN`...`ELSE`...`END IF`, `SELECT`...`CASE`...`END SELECT`, `WHILE`...`WEND`, `DEF FN`...`END DEF`, `FUNCTION`...`END FUNCTION`, and `SUB`...`END SUB`), in the error-handling routine, event-handling routine, or in the subroutines.

Syntax errors:

Error code and message	Meaning
`error 64: Function redefinition`	You made double definition to a same function name.
`error 71: Syntax error`	• *`stringlength`* is out of the range. • *`stringlength`* is not an integer constant.

Reference:

Statements: `FUNCTION`...`END FUNCTION` and `SUB...END SUB`

DEF FN (Single-line form)

Names and defines a user-created function.

Syntax:

Syntax 1 (Defining a numeric function):

```
DEF FNfunctionname[(dummyparameter[,dummyparameter
...])]=expression
```

Syntax 2 (Defining a string function):

```
DEF FNfunctionname[(dummyparameter
[,dummyparameter...])] [[stringlength]]=expression
```

Syntax 3 (Calling the function):

```
FNfunctionname[(realparameter[,realparameter ...])]
```

Parameter:

functionname

- For numerics

functionname%	Integer function name
functionname	Real function name

- For strings

functionname$	Character function name

where the FN can be in lowercase.

dummyparameter

A non-array integer variable, a non-array real variable, or a non-array string variable.

stringlength

An integer constant having a value from 1 to 255.

expression and *realparameter*

A numeric or string expression.

Description:

■ Creating a user-defined function

`DEF FN` creates a user-defined function.

- Definition of a user-defined function should appear preceding a calling statement of the user-defined function in a source program.
- You cannot make double definition to a same function name.
- The `DEF FN` statement should not be defined in the block-structured statements (`DEF FN`...`END DEF`, `FOR`...`NEXT`, `FUNCTION`...`END FUNCTION`, `IF`...`THEN` ...`ELSE`...`END IF`, `SELECT`...`CASE`...`END SELECT`, `SUB`...`END SUB` and `WHILE`...`WEND`), in the error-handling routine, event-handling routine, or in the subroutines.
- `DEF FN` functions cannot be recursive.
- The type of *`functionname`* should match that of the function definition *`expression`*.
- In defining a character function, you can specify the maximum *`stringlength`* for a return value. If its specification is omitted, the default value of 40 characters takes effect.
- *`dummyparameter`*, which corresponds to the variable having the same name in the function definition *`expression`*, is a local variable valid only in that *`expression`* . Therefore, if a variable having the same name as *`dummypa-rameter`* is used outside `DEF FN` statement or used as a *`dummyparameter`* of any other function in the same program, it will be independently treated.
- *`expression`* describes some operations for the user-defined function. It should be within one program line including definition described left to the equal sign.
- *`expression`* can call other user-defined functions. You can nest `DEF FN` statements to a maximum of 10 levels.
- If variables other than *`dummyparameter`*(s) are specified in *`expression`*, they will be treated as global variables whose current values are available.
- *`stringlength`* should be enclosed with a pair of square brackets [ ].

■ Calling a user-defined function

`FN`*`functionname`* calls a user-defined function.

- The number of *`realparameter`*s should be equal to that of *`dummyparame-ter`*s, and the types of the corresponding variables used in those parameters should be identical.
- If you specify a global variable in *`realparameter`* when calling a user-defined function, the user-defined function cannot update the value of the global variable. This is because all *`realparameter`*s are passed not by address but by value. (So called "Call-by-value")

Syntax errors:

■ When defining a user-defined function

Error code and message	Meaning
`error 61: Cannot use DEF FN in control structure`	The `DEF FN` statement is defined in block-structured statements such as `FOR` and `IF` statements.
`error 64: Function redefinition`	You made double definition to a same function name.
`error 65: Function definitions exceed 200`	———
`error 66: Arguments exceed 50`	———
`error 71: Syntax error`	• *`functionname`* is an integer function name, but *`expression`* is a real type. (If *`functionname`* is a real function name and *`expression`* is an integer type, then no error occurs.) • *`stringlength`* is out of the range. • *`stringlength`* is not an integer constant.

■ When calling a user-defined function

Error code and message	Meaning
`error 68: Mismatch argument type or number`	• The number of the real parameters is not equal to that of the dummy parameters. • *`dummyparameter`* was an integer variable in defining a function, but *`realparameter`* is a real type in calling the function. (If *`dummyparameter`* was a real variable in defining a function and *`realparameter`* is an integer type, then no error occurs.)
`error 69: Function undefined`	Calling of a user-defined function precedes the definition of the user-created function.

Run-time errors:

Error code	Meaning
`07h`	Insufficient memory space (You nested `DEF FN` statements to more than 10 levels.)
`0Fh`	String length out of the range (The returned value of the *`stringlength`* exceeds the allowable range.)

Example:

- Example 1

```
DEF FNadd(a%,b%)=a%+b%
PRINT FNadd(3,5)
```

```
8
```

- Example 2

```
DEF FNappend$(a$,b$)[80]=a$+b$
PRINT FNappend$("123","AB")
```

```
123AB
```

DEF FN...END DEF (Block form)

Names and defines a user-created function.

Syntax:

Syntax 1 (Defining a numeric function):

DEF FN*functionname*[(*dummyparameter*[,*dummyparameter* ...])]

Syntax 2 (Defining a character function):

DEF FN*charafunctionname*[(*dummyparameter* [,*dummyparame-ter*...])] [[*stringlength*]]

Syntax 3 (Exiting from the function block prematurely):

```
EXIT DEF
```

Syntax 4 (Ending the function block):

```
END DEF
```

Syntax 5 (Assigning a returned value):

FN*functionname* = *generalexpression*

Syntax 6 (Calling a function):

FN*functionname*[(*realparameter*[,*realparameter* ...])]

Parameter:

Same as for `DEF FN` (Single-line form).

Description:

■ Creating a user-defined function

`DEF FN`...`END DEF` creates a user-defined function. The function definition block between `DEF FN` and `END DEF` is a set of some statements and functions.

- Definition of a user-defined function should appear preceding a calling statement of the user-defined function in a source program.
- You cannot make double definition to a same function name.
- This statement block should not be defined in the block-structured statements (`DEF FN`...`END DEF`, `FOR`...`NEXT`, `FUNCTION`...`END FUNCTION`, `IF`...`THEN`...`ELSE`...`END IF`, `SELECT`...`CASE`...`END SELECT`, `SUB`...`END SUB` and `WHILE` ...`WEND`), in the error-handling routine, event-handling routine, or in the subroutines.
- `DEF FN`...`END DEF` functions can be recursive.
- In defining a character function, you can specify the maximum *`stringlength`*. If its specification is omitted, the default value of 40 characters takes effect.
- *`dummyparameter`*, which corresponds to the variable having the same name in the function definition block, is a local variable valid only in that block. Therefore, if a variable having the same name as *`dummyparameter`* is used outside `DEF FN`...`END DEF` statement block or used as a *`dummyparameter`* of any other function in the same program, it will be independently treated.
- In user-defined functions, you can call other user-defined functions. You can nest `DEF FN`...`END DEF` statements to a maximum of 10 levels.
- When using the `DEF FN`...`END DEF` together with block-structured statements (`DEF FN`...`END DEF`, `FOR`...`NEXT`, `FUNCTION`...`END FUNCTION`, `IF`...`THEN`...`ELSE`...`END IF`, `SELECT`...`CASE`...`END SELECT`, `SUB`...`END SUB` and `WHILE` ...`WEND`), you can nest them to a maximum of 30 levels.
- If variables other than *`dummyparameter`*(s) are specified in the function definition block, they will be treated as global variables whose current values are available.
- `EXIT DEF` exits the function block prematurely and returns control to the position immediately after the statement that called the user-defined function.
- The block-format `DEF FN` statement should be followed by `END DEF` which ends the function block and returns control to the position immediately after the statement that called the user-defined function.
- Using Syntax 5 allows you to assign a return value for a function. The type of *`functionname`* should match that of a return value. If no return value is assigned to *`functionname`* , the value 0 or a null string will be returned for a numeric function or a character function, respectively.

■ Calling a user-defined function

`FN`*`functionname`* calls a user-defined function.

- The number of *`realparameter`*s should be equal to that of *`dummyparameter`*s, and the types of the corresponding variables used in those parameters should be identical.
- If you specify a global variable in *`realparameter`* when calling a user-defined function, the user-defined function cannot update the value of the global variable. This is because all *`realparameter`*s are passed not by address but by value. (So called "Call-by-value")

Syntax errors:

■ When defining a user function

Error code and message		Meaning
`error 59:`	`Incorrect use of DEF FN... EXIT DEF...END DEF`	• The `EXIT DEF` statement is specified outside the function definition block. • The `END DEF` statement is specified outside the function definition block.
`error 60:`	`Incomplete control structure (DEF FN ...END DEF)`	`END DEF` is missing.
`error 61:`	`Cannot use DEF FN in control structure`	The `DEF FN...END DEF` statement is defined in other block-structured statements such as `FOR` and `IF` statement blocks.
`error 64:`	`Function redefinition`	You made double definition to a same function name.
`error 71:`	`Syntax error`	• *`functionname`* is an integer function name, but *`generalexpression`* is a real type. (If *`functionname`* is a real function name and *`generalexpression`* is an integer type, then no error occurs.) • *`stringlength`* is out of the range. • *`stringlength`* is not an integer constant. • The function name is assigned a value outside the function definition block.

■ When calling a user-defined function

Error code and message	Meaning
`error 68: Mismatch argument type or number`	• The number of the real parameters is not equal to that of the dummy parameters. • *dummyparameter* was an integer variable in defining a function, but *realparameter* is a real type in calling the function. (If *dummyparameter* was a real variable in defining a function and *realparameter* is an integer type, then no error occurs.)
`error 69: Function undefined`	Calling of a user-defined function precedes the definition of the function.

Run-time errors:

Error code	Meaning
07h	Insufficient memory space (You nested DEF FN statements to more than 10 levels.)
0Dh	END DEF out of the DEF FN block
0Fh	String length out of the range (The returned value of *stringlength* exceeds the allowable range.)

Example:

```
DEF FNappend$(a$,b%)[128]
   C$=""
   FOR i%=1 TO b%
      C$=C$+a$
   NEXT
   FNappend$=C$
END DEF
PRINT FNappend$("AB",3)
```

```
ABABAB
```

DEFREG

Defines register variables.

Syntax:

DEFREG *registerdefinition*[,*registerdefinition* ...]

Parameter:

registerdefinition

non-arraynumericvariable [=*numericconstant*]

```
DEFREG n1%=10
DEFREG n2=12.5
```

arraynumericvariable(*subscript*)
[=*numericinitialvaluedefinition*]

```
DEFREG n3(5,6)
```

non-arraystringvariable[[*stringlength*]]
[=*stringconstant*]

```
DEFREG s1$="abc123"
DEFREG s2$[6]="abc123"
```

arraystringvariable(*subscript*)[[*stringlength*]]
[=*stringinitialvaluedefinition*]

```
DEFREG s2$(1,3)[16]
```

subscript

For one-dimensional:*integerconstant*

```
DEFREG n4%(3)
```

For two-dimensional:*integerconstant*,*integerconstant*

```
DEFREG n5%(4,5)
```

Where *integerconstant* is a value from 0 to 254.

numericinitialvaluedefinition

For one-dimensional:
`{`*numericconstant*`[,`*numericconstant*`...]}`

```
DEFREG n6%(3)={9,8,7,6}
```

For two-dimensional:
`{{`*numericconstant*`[,`*numericconstant*`...]}, {`*numericconstant*`[,`*numericconstant*`...]} ...}`

```
DEFREG n7(1,2)={{10,11,12},{13,14,15}}
```

stringinitialvaluedefinition

For one-dimensional:
`{`*stringconstant*`[,`*stringconstant*`...]}`

```
DEFREG s3$(3)={"a","bc","123","45"}
```

For two-dimensional:
`{{`*stringconstant*`[,`*stringconstant*`...]}, {`*stringconstant*`[,`*stringconstant*`...]} ...}`

```
DEFREG s4$(1,1)={{"a","b"},{"c","1"}}
```

stringlength

An integer constant from 1 to 255.

Description:

`DEFREG` defines non-array or array register variables.

- A `DEFREG` statement can appear anywhere in a source program.
- Up to 2-dimensional array variables can be defined.
- For both *non-arraystringvariable* and *arraystringvariable*, the string length can be specified.
- Defaults:

stringlength for non-array variables:	40 characters
stringlength for array variables:	20 characters

- The memory area for register variables is allocated in user program files in the memory. Register variables, therefore, are always updated. An uploaded user program, for example, contains the updated register variables if defined.
- The total number of bytes allowable for register variables is 64 kilobytes.
- You can specify an initial value to an array variable by enclosing it with a pair of braces { }. No comma (,) is allowed for terminating the list of initial values.

  If the number of the specified initial values is less than that of the array elements or if no initial value is specified, then the Compiler automatically sets a zero (0) or a null string as an initial value for a numeric variable or a string variable of the array elements not assigned initial values, respectively.

Syntax errors:

Error code and message	Meaning
`error 6: Variable name redefinition`	A same register variable name is double declared in a program.
`error 71: Syntax error`	• *`stringlength`* is not an integer constant. • The number of the specified initial values is greater than that of the array elements. • The list of initial values is terminated with a comma. • The type of the specified variable does not match that of its initial value. (Note that a real variable can have an integer constant as an initial value.) • *`subscript`* is not an integer constant.
`error 73: Improper string length`	*`stringlength`* is out of the range.
`error 74: Improper array element number`	*`subscript`* is out of the range.
`error 75: Out of space for register variable area`	Definition by `DEFREG` exceeds the register variable area.
`error 77: Initial string too long`	• The dimension of the specified array variable does not match that of its initial value. • The number of initial value elements for the specified register string variable is greater than its string length.
`error 83: ')' missing`	No closing parenthesis follows *`subscript`*.
`error 84: ']' missing`	No closing square bracket follows *`stringlength`*.
`error 90: '{' missing`	No opening brace precedes the initial value.

Example:

Example 1: Valid `DEFREG` statements

```
DEFREG a,e$
DEFREG b=100,c(10),d$(2,4)[10]
DEFREG bps$="19200"
DEFREG a%(2)={1,2}
DEFREG a%(2)={1,,3}
DEFREG a%(2)={,,3}
DEFREG b%(1,1)={{},{1,2}}
DEFREG b%(1,1)={,{1,2}}
DEFREG b%(1,1)={{1,2}}
```

Example 2: Position of elements in an array

```
DEFREG a%(1,1)={{1},{,3}}
```

The elements of the above array have the following initial values:

```
a%(0,0):1
a%(0,1):0
a%(1,0):0
a%(1,1):3
```

```
DEFREG b$(1,1)[3]={,{"123"}}
```

The elements of the above array have the following initial values:

```
b$(0,0):""
b$(0,1):""
b$(1,0):"123"
b$(1,1):""
```

Example 3: `DEFREG` statements causing syntax errors

```
DEFREG c%(2)={1,2,3,4}
DEFREG d%(2)={1,2,}
DEFREG e%(1,1)={{,},{1,2}}
DEFREG f%(1,1)={{1,2},}
```

Reference:

Statements: `DIM`

DIM

Declares and dimensions arrays; also declares the string length for a string variable.

Syntax:

`DIM` *`arraydeclaration`*`[,`*`arraydeclaration`*`...]`

Parameter:

`arraydeclaration`

`numericvariable` `(`*`subscript`*`)`

```
DIM n1%(12)
DIM n2(5,6)
```

`stringvariable` `(`*`subscript`*`)[[`*`stringlength`*`]]`

```
DIM s1$(2)
DIM s2$(2,6)
DIM s3$(4)[16]
DIM s4$(5,3)[30]
```

`subscript`

For one-dimensional:	*`integerexpression`*
For two-dimensional:	*`integerexpression`*, *`integerexpression`*

Where *`integerexpression`* is a numeric expression which returns a value from 0 to 254.

`stringlength`

An integer constant which has a value from 1 to 255 which indicates the number of characters.

Description:

`DIM` declares array variables and dimensions the arrays that a program will utilize.

- A `DIM` statement can appear anywhere before the first use of the array in a source program. However, when possible, you should place all your `DIM` statements together near the beginning of the program and should not place them in the program execution loops in order to prevent errors.
- Up to 2-dimensional array variables can be declared.

- In declaring an array string variable, you can specify the string length. If its specification is omitted, the default value of 20 characters takes effect.
- If no subscript is specified for a string variable, the Compiler automatically regards the string variable as a non-array string variable so that the default for a non-array string variable, 40 characters, takes effect.

Syntax errors:

Error code and message		Meaning
`error 7:`	`Variable name redefinition`	The array declared with `DIM` had been already declared with `DEFREG`.
`error 71:`	`Syntax error`	• *`stringlength`* is out of the range. • *`stringlength`* is not an integer constant.
`error 72:`	`Variable name redefinition`	• A same variable name is double declared inside a same `DIM` statement. • A same variable name is used for a non-array variable and array variable.
`error 78:`	`Array symbols exceed 30 for one DIM statement`	More than 30 variables are declared inside one `DIM` statement.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range
`07h`	Insufficient memory space (The variable area has run out.)
`0Ah`	Duplicate definition (An array is double declared.)

Reference:

Statements: `DEFREG` and `ERASE`

END

Terminates program execution.

Syntax:

```
END
```

Description:

`END` terminates program execution and sounds the beeper for a second.

- An `END` can appear anywhere in a source program.
- When an `END` statement occurs, all of the files being opened become closed, and then the following operation takes place depending upon whether or not any application program (user program or Easy Pack) has been selected as an execution program (to be run when the BHT is powered on) in System Mode.
  - If any application program has been selected, the BHT turns off the power after three seconds from the message indication of the "Program end."
  - If an execution program has not been selected, control passes to System Mode.

  (For System Mode, refer to the BHT User's Manual.)

Memory control statement

ERASE

Erases array variables.

Syntax:

`ERASE` *`arrayvariablename`*`[,`*`arrayvariablename`*`...]`

Parameter:

`arrayvariablename`

An array numeric or array string variable.

Description:

ERASE erases an array variable(s) specified by *`arrayvariablename`* and frees the memory used by the array.

- *`arrayvariablename`* is the name of an array variable already declared by the `DIM` statement. If it has not been declared by `DIM`, the ERASE statement will be ignored.
- After erasing the name of an array variable with `ERASE`, you can use that name to declare a new array variable with the `DIM` statement.
- *`arrayvariablename`* should not include subscripts or parentheses ( ) as shown below.

```
DIM a(3),b1%(5,10),c$(3)[20]
ERASE a,b1%,c$
```

- `ERASE` cannot erase a register variable declared by the `DEFREG` statement, a common variable declared by the `COMMON` statement, or a non-array string variable.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	You attempted to erase a register variable declared by `DEFREG`, a common variable by `COMMON`, or a non-array string variable.

Reference:

Statements:	`DEFREG` and `DIM`

FIELD

Allocates string variables as field variables.

Syntax:

```
FIELD [#]filenumber,fieldwidth AS fieldvariable
[,fieldwidth AS fieldvariable...]
```

Parameter:

filenumber

A numeric expression which returns a value from 1 to 16.

fieldwidth

A numeric expression which returns a value from 1 to 254.

fieldvariable

A non-array string variable.

Description:

`FIELD` declares the length and field variable of each field of a record in a data file.

- *filenumber* is the file number of a data file opened by the `OPEN` statement.
- *fieldwidth* is the number of bytes for a corresponding field variable.
- You can assign a same field variable to more than one field.
- There is no difference in usage between a field variable and a general variable except that no register variable, common variable, or array variable can be used for a field variable.
- A record can contain up to 16 fields. The total number of bytes of all *fieldwidth*s plus the number of fields should not exceed 255.
- If a `FIELD` statement executes for an opened file having the number of fields or field width unmatching that of the *FIELD* specifications except for field variables, a run-time error will occur.
- If more than one *FIELD* statement is issued to a same file, the last one takes effect.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	*`filenumber`* is missing.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`fieldwidth`* out of the range)
`34h`	Bad file name or number (You specified *`filenumber`* of an unopened file.)
`36h`	Improper file type (You specified *`filenumber`* of a file other than data files.)
`3Ah`	File number out of the range
`3Ch`	`FIELD` overflow (A `FIELD` statement specifies the record length exceeding 255 bytes.)
`3Dh`	A `FIELD` statement specifies the field width which does not match one that specified in file creation.

Example:

```
fileNumber% = 4
OPEN "Datafile.dat" AS #fileNumber%
FIELD #fileNumber%,20 AS code39$,
16 AS itf$,5 AS kyin$
```

Reference:

Statements: `CLFILE`, `CLOSE`, `GET`, `OPEN` and `PUT`

FOR...NEXT

Defines a loop containing statements to be executed a specified number of times.

Syntax:

FOR *controlvariable* = *initialvalue* TO *finalvalue* [STEP *increment*]

⋮

NEXT [*controlvariable*]

Parameter:

controlvariable

A non-array numeric variable.

initialvalue, *finalvalue*, and *increment*

Numeric expressions.

Description:

FOR...NEXT defines a loop containing statements (which is called "body of a loop") to be executed by the number of repetitions controlled by *initialvalue*, *finalvalue*, and *increment*.

■ Processing procedures

(1) The Interpreter assigns *initialvalue* to *controlvariable*.

(2) The Interpreter checks terminating condition; that is, it compares the value of *controlvariable* against the *finalvalue*.

- When the value of *increment* is positive:

If the value of *controlvariable* is equal to or less than the *finalvalue*, go to step (3). If it becomes greater the *finalvalue*, the program proceeds with the first line after the NEXT statement (the loop is over).

- When the value of *increment* is negative:

If the value of *controlvariable* is equal to or greater than the *finalvalue*, go to step (3). If it becomes less than the *finalvalue*, the program proceeds with the first line after the NEXT statement (the loop is over).

(3) The body of the loop executes and the NEXT statement increases the value of *controlvariable* by the value of *increment*. Then, control returns to the FOR statement at the top of the loop. Go back to step (2).

- The default value of *`increment`* is 1.
- You can nest `FOR`...`NEXT` statements to a maximum of 10 levels.
- When using the `FOR`...`NEXT` statement together with block-structured statements (`DEF FN`...`END DEF`, `FOR`...`NEXT`, `FUNCTION`...`END FUNCTION`, `IF`...`THEN` ...`ELSE`...`END IF`, `SELECT`...`CASE`...`END SELECT`, `SUB`...`END SUB` and `WHILE`...`WEND`), you can nest them to a maximum of 30 levels.
- A same *`controlvariable`* should not be reused in a nested loop. Otherwise, a run-time error will occur when the `NEXT` statement for an outer `FOR`...`NEXT` loop executes.
- Nested loops should not be crossed. Shown below is a correctly nested sample.

```
FOR i%=1 TO 10
    FOR j%=2 TO 100
        FOR k%=3 TO 1000
        NEXT k%
    NEXT j%
NEXT i%
FOR l%=1 TO 3
  ⋮
NEXT l%
```

Syntax errors:

Error code and message		Meaning
`error 26:`		Too deep nesting.
`error 52:`	`Incorrect use of FOR…NEXT`	`NEXT` without `FOR`.
`error 53:`	`Incomplete control structure`	Incomplete pairs of `FOR` and `NEXT`.
`error 54:`	`Incorrect FOR index variable`	*`controlvariable`* for `FOR` is different from that for `NEXT`.
`error 88:`	`'TO' missing`	`TO` *`finalvalue`* is missing.

Run-time errors:

Error code	Meaning
`01h`	`NEXT` without `FOR`
`07h`	Insufficient memory space (Too deep nesting.)

FUNCTION…END FUNCTION

Names and defines user-created function FUNCTION.

Syntax:

Syntax 1 (Defining a numeric function):

```
FUNCTION funcname [(dummyparameter
[,dummyparameter...])]
```

Syntax 2 (Defining a character function):

```
FUNCTION funcname [(dummyparameter
[,dummyparameter...])][[stringlength]]
```

Syntax 3 (Existing from the function block prematurely):

```
EXIT FUNCTION
```

Syntax 4 (Ending the function block):

```
END FUNCTION
```

Syntax 5 (Assigning a returned value):

```
funcname = generalexpression
```

Syntax 6 (Calling a function):

```
funcname[(realparameter[,realparameter...])]
```

Parameter:

funcname

- For numerics

funcname%	Integer function name
funcname	Real function name

- For strings

funcname$	Character function name

dummyparameter

A non-array integer variable, a non-array real variable, or a non-array string variable.

stringlength

An integer constant having a value from 1 to 255.

realparameter

A numeric or string expression.

Description:

■ Creating a user-defined function

`FUNCTION`...`END FUNCTION` creates a user-defined function. The function definition block between `FUNCTION` and `END FUNCTION` is a set of some statements and functions.

- You cannot make double definition to a same function name.
- This statement block should not be defined in the block-structured statements (`DEF FN`...`END DEF`, `FOR`...`NEXT`, `FUNCTION`...`END FUNCTION`, `IF`...`THEN` ...`ELSE`...`END IF`, `SELECT`...`CASE`...`END SELECT`, `SUB`...`END SUB` and `WHILE`...`WEND`), in the error-handling routine, event-handling routine, or in the subroutines.
- `FUNCTION`...`END FUNCTION` functions can be recursive.
- In defining a character function, you can specify the maximum *`stringlength`*. If its specification is omitted, the default value of 40 characters takes effect.
- *`dummyparameter`*, which corresponds to the variable having the same name in the function definition block, is a local variable valid only in that block. Therefore, if a variable having the same name as *`dummyparameter`* is used outside `FUNCTION`...`END FUNCTION` statement block or used as a *`dummyparameter`* of any other function in the same program, it will be independently treated.
- In user-defined functions, you can call other user-defined functions. You can nest `FUNCTION`...`END FUNCTION` statements to a maximum of 10 levels.
- When using the `FUNCTION`...`END FUNCTION` together with block-structured statements (`DEF FN`...`END DEF`, `FOR`...`NEXT`, `FUNCTION`...`END FUNCTION`, `IF`...`THEN`...`ELSE`...`END IF`, `SELECT`...`CASE`...`END SELECT`, `SUB`...`END SUB` and `WHILE`...`WEND`), you can nest them to a maximum of 30 levels.
- If variables other than *`dummyparameter`*(s) are specified in the function definition block, they will be treated as local variables whose current values are available only in that function definition block, unless PRIVATE or GLOBAL is specified.
- `EXIT FUNCTION` exits the function block prematurely and returns control to the position immediately after the statement that called the user-defined function.
- Using Syntax 5 allows you to assign a return value for a function. The type of *`funcname`* should match that of a return value. If no return value is assigned to *`funcname`* , the value 0 or a null string will be returned for a numeric function or a character function, respectively.

■ Calling a user-defined function

`funcname` calls the function.

- The number of *`realparameter`*s should be equal to that of *`dummyparameter`*s, and the types of the corresponding variables used in those parameters should be identical.

- If you specify a global variable in *`realparameter`* when calling a user-defined function, the user-defined function cannot update the value of the global variable. This is because all *`realparameter`*s are passed not by address but by value. (So called "Call-by-value")

NOTE

Before any call to a `FUNCTION...END FUNCTION`, you need to place definition of the `FUNCTION` function or declaration of the `FUNCTION` by the `DECLARE` statement in your source program.

A function name is defined globally. If more than one same function name exists in a same project, therefore, a multiple symbol definition error will occur when files will be linked. The same error will occur also if the `FUNCTION...END FUNCTION` defines a user-created function in a file to be included and more than one file in a same project reads that included file.

Syntax errors:

■ When defining a user function

Error code and message	Meaning
`error 64: Function redefinition`	You made double definition to a same function name.
`error 71: Syntax error`	• *`funcname`* is an integer function name, but *`generalexpression`* is a real type. (If *`funcname`* is a real function name and *`generalexpression`* is an integer type, then no error occurs.) • *`stringlength`* is out of the range. • *`stringlength`* is not an integer constant. • The function name is assigned a value outside the function definition block.
`error 95: Incorrect use of FUNCTION, EXIT FUNCTION, or END FUNCTION`	• The `EXIT FUNCTION` statement is specified outside the function definition block. • The `END FUNCTION` statement is specified outside the function definition block.

Error code and message	Meaning
`error 96: Incomplete control structure (FUNCTION...END FUNCTION)`	`END FUNCTION` is missing.
`error 97: Cannot use FUNCTION in control structure`	The `FUNCTION`...`END FUNCTION` statement is defined in other block-structured statements such as `FOR` and `IF` statement blocks.

■ When calling a user-defined function

Error code and message	Meaning
`error 68: Mismatch argument type or number`	• The number of the real parameters is not equal to that of the dummy parameters. • *`dummyparameter`* was an integer variable in defining a function, but *`realparameter`* is a real type in calling the function. (If *`dummyparameter`* was a real variable in defining a function and *`realparameter`* is an integer type, then no error occurs.)
`error 69: Function undefined`	Calling of a user-defined function precedes the definition of the user-defined function.

Run-time errors:

Error code	Meaning
`07h`	Insufficient memory space (You nested `FUNCTION` statements to more than 10 levels.)
`0Fh`	String length out of the range (The returned value of *`stringlength`* exceeds the allowable range.)

Example:

File 1

```
DECLARE FUNCTION add(x,y)
A=1:B=2
PRINT "TEST"
C=add(A,B)
PRINT C
   ⋮
```

File 2

```
FUNCTION  add(X,Y)
  add=X+Y
END FUNCTION
```

```
TEST
3
```

Reference:

Statements: DECLARE

File I/O statement

GET

Reads a record from a data file.

Syntax:

```
GET [#]filenumber[,recordnumber]
```

Parameter:

filenumber

A numeric expression which returns a value from 1 to 16.

recordnumber

A numeric expression which returns a value from 1 to 32767.

Description:

`GET` reads the record specified by *recordnumber* from the data file specified by *filenumber* and assigns the data to the field variable(s) specified by the `FIELD` statement.

- *filenumber* is the file number of a data file opened by the `OPEN` statement.
- If a data file having no record is specified, a run-time error will occur.
- The first record in a data file is counted as 1.
- If no *recordnumber* is specified, the `GET` statement reads a record whose number is one greater than that of the record read by the preceding `GET` statement.

  If no *recordnumber* is specified in the first `GET` statement after opening of a file, the first record (numbered 1) in the file will be read.
- *recordnumber* should be equal to or less than the number of written records. If it is greater, a run-time error will occur.
- If a `GET` statement without *recordnumber* is executed after occurrence of a run-time error caused by an incorrect record number in the preceding `GET` statement, then the new `GET` statement reads the record whose record number is one greater than that of the latest record correctly read.
- If a `GET` statement without *recordnumber* is executed after execution of the preceding `GET` statement specifying the last record (the number of the written records), then a run-time error will occur.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	*`filenumber`* is missing.

Run-time errors:

Error code	Meaning
`34h`	Bad file name or number (You specified *`filenumber`* of an unopened file.)
`36h`	Improper file type (You specified *`filenumber`* of a file other than data files.)
`3Ah`	File number out of the range
`3Eh`	A `PUT` or `GET` statement executed without a `FIELD` statement.
`3Fh`	Bad record number (No record to be read in a data file.)

Example:

```
GET #filNo,RecordNo
GET #4
GET #3,100
```

Reference:

Statements: `FIELD`, `OPEN`, and `PUT`

Declarative statement

GLOBAL

Declares one or more work variables or register variables defined in a file, to be global.

Syntax:

`GLOBAL` *`varname`* `[,`*`varname`*`...]`

Parameter:

`varname`

`numericvariable` `[(`*`subscript`*`)]`

`stringvariable` `[(`*`subscript`*`)[`*`stringlength`*`]]`

`subscript`

For one-dimensional:	*`integerconstant`*
For two-dimensional:	*`integerconstant`*, *`integerconstant`*

Where *`integerconstant`* is a numeric expression which returns a value from 0 to 254.

`stringlength`

An integer constant from 1 to 255.

Description:

`GLOBAL` allows variables declared by *`varname`* to be referred to or updated in other programs.

- If a same variable name as specified inside the `GLOBAL` statement is already declared in your file, the `GLOBAL` statement will result in an error.
- Up to 30 variables can be declared inside one `GLOBAL` statement.
- You may declare non-array variables and array variables together inside one `GLOBAL` statement.

Syntax errors:

Error code and message	Meaning
`error 7:` `Variable name redefinition`	The array declared with `GLOBAL` statement had been already declared with `DEFREG` statement.
`error 71:` `Syntax error`	• *`stringlength`* is out of the range. • *`stringlength`* is not an integer constant.
`error 72:` `Variable name redefinition`	• A same variable name is double declared inside a same `GLOBAL` statement. • A same variable name is used for a non-array variable and array variable.
`error 78:` `Array symbols exceed 30 for one DIM, PRIVATE, or GLOBAL statement`	• More than 30 variables are declared inside one `GLOBAL` statement.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range
`07h`	Insufficient memory space (The variable area has run out.)
`0Ah`	Duplicate definition (An array is double declared.)

Reference:

Statements: `DIM` and `PRIVATE`

Flow control statement

GOSUB

Branches to a subroutine.

Syntax:

```
GOSUB label
```

Description:

`GOSUB` calls a subroutine specified by *`label`*.

- Within the subroutine itself, you use a `RETURN` statement which indicates the logical end of the subroutine and returns control to the statement just after the `GOSUB` that called the subroutine.
- You may call a subroutine any number of times as long as the Interpreter allows the nest level and other conditions.
- Subroutines can appear anywhere in a source program. However, you should separate subroutines from the main program by any means such as by placing subroutines immediately following the `END` or `GOTO` statement, in order to prevent the main part of the program from falling into those subroutines.
- A subroutine can call other subroutines. You can nest `GOSUB` statements to a maximum of 10 levels.
- When using the `GOSUB` statement together with block-structured statements (`DEF FN`...`END DEF`, `FOR`...`NEXT`, `FUNCTION`...`END FUNCTION`, `IF`...`THEN` ...`ELSE`...`END IF`, `SELECT`...`CASE`...`END SELECT`, `SUB`...`END SUB`, and `WHILE`...`WEND`), you can nest them to a maximum of 30 levels.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• *`label`* has not been defined. • *`label`* is missing.

Run-time errors:

Error code	Meaning
`03h`	`RETURN` without `GOSUB` statement
`07h`	Insufficient memory space (Too deep nesting)

Reference:

Statements: `RETURN`

Flow control statement

GOTO

Branches to a specified label.

Syntax:

```
GOTO label
```

Description:

GOTO unconditionally transfers control to a label specified by *label*.

- In an IF statement block, you can omit GOTO immediately following THEN or ELSE, as shown below.

  ```
  IF a=0 THEN Lbl1 ELSE Lbl2
  END IF
  ```

- GOTO allows you to branch anywhere in your program. However, you should branch only to another line in a program module or subroutine at the same program level. Avoid transferring control to a DEF FN block or other blocks at the different program level.
- You can use GO TO instead of GOTO.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• *label* has not been defined. • *label* is missing.

IF...THEN...ELSE...END IF

Conditionally executes specified statement blocks depending upon the evaluation of a conditional expression.

Syntax:

Syntax 1:

```
IF conditionalexpression THEN
     statementblock1
[ELSE
     statementblock2]
END IF
```

Syntax 2:

```
IF conditionalexpression ELSE
     statementblock
END IF
```

Parameter:

conditionalexpression

A numeric expression which evaluates to true or false.

Description:

`IF` statement block tests whether *conditionalexpression* is true or false. If the condition is true (not zero), *statementblock* which follows `THEN` is executed; if it is false (zero), *statementblock* which follows `ELSE` is executed. Then, program control passes to the first statement after `END IF`.

- You can omit either `THEN` block or `ELSE` block.
- `IF` statement block should terminate with `END IF` which indicates the end of the block.
- `IF` statement blocks can be nested. When using the `IF` statement block together with other block-structured statements (`DEF FN`...`END DEF`, `FOR`...`NEXT`, `FUNCTION`...`END FUNCTION`, `IF`...`THEN`...`ELSE`...`END IF`, `SELECT`...`CASE`...`END SELECT`, `SUB`...`END SUB`, and `WHILE`...`WEND`), you can nest them to a maximum of 30 levels.

- A block-structured `IF` statement block has the following advantages over a single-line `IF` statement (which is not supported in BHT-BASIC):
  - More complex conditions can be tested since an `IF` statement block can contain more than one line for describing conditions.
  - You can describe as many statements or statement blocks as you want.
  - Since it is not necessary to put more than one statement in a line, you can describe easy-to-read programs according to the logical structure, making correction and debugging easy.
- You can use `ENDIF` instead of `END IF`.

Syntax errors:

Error code and message		Meaning
`error 26:`		Too deep nesting.
`error 50:`	`Incorrect use of IF...THEN ...ELSE...END IF`	`THEN` is missing.
`error 51:`	`Incomplete control structure`	`END IF` is missing.

Example:

```
k$=INKEY$
IF k$<>"" THEN
      PRINT k$;
END IF
```

Reference:

Statements:	`DEF FN...END DEF`, `FOR...NEXT`, `ON...GOSUB`, `ON...GOTO`, `SELECT...CASE...END SELECT`, and `WHILE...WEND`

INPUT

Reads input from the keyboard into a variable.

Syntax:

`INPUT [;]["`*`prompt`*`"{,|;}]`*`variable`*

Parameter:

"*prompt*"
: A string constant.

variable
: A numeric or string variable.

Description:

When execution reaches an `INPUT` statement, the program pauses and waits for the user to enter data from the keyboard while showing a prompting message specified by "*prompt*".

After typing data, the user must press the ENT key. Then, the `INPUT` statement assigns the typed data to *variable*.

- "*prompt*" is a prompting message to be displayed on the LCD.
- The semicolon (;) or comma (,) after "*prompt*" has the following meaning:

  If "*prompt*" is followed by a semicolon, the `INPUT` statement displays the prompting message followed by a question mark and a space.

  ```
  INPUT "data= ";a$
  ```

  ```
  data= ?
  ```

  If "*prompt*" is followed by a comma, the statement displays the prompting message but no question mark or space is appended to the prompting message.

  ```
  INPUT "data= ",a$
  ```

  ```
  data=
  ```

- The cursor shape specified by the most recently executed `LOCATE` statement takes effect.

- Even after execution of the `CURSOR OFF` statement, the `INPUT` statement displays the cursor.
- Data inputted by the user will echo back to the LCD. To assign it to *`variable`*, it is necessary to press the ENT key.

  Pressing the ENT key causes also a line feed. If `INPUT` is followed by a semicolon (;) in an `INPUT` statement, however, line feed is suppressed.

  If you type no data and press the ENT key, an `INPUT` statement automatically assigns a zero or a null string to *`variable`* that is a numeric or string, respectively.
- When any echoed back data is displayed on the LCD, pressing the Clear or BS key erases the whole displayed data or a most recently typed-in character of the data, respectively. If no data is displayed, pressing the Clear orþBSþkey produces no operation.
- Notes for entering numeric data:

  The effective length of numeric data is 12 characters. The 13th typed-in literal and the following will be ignored.

  Valid literals include 0 to 9, a minus sign (-), and a period (.). They should be in correct numeric data form. If not, `INPUT` statement accepts only numeric data from the first literal up to correctly formed literal, as valid data. If no valid data is found, the `INPUT` statement automatically assigns a zero (0) to *`variable`*.

  A plus sign (+) can be typed in, but it will be ignored in evaluation of the typed-in data.
- Notes for entering string data:

  The effective length of string data is the maximum string length of *`variable`*. Overflowed data will be ignored.
- The sizes of prompting message literals, echoed back literals and cursor depend upon the screen mode (the single-byte ANK mode, two-byte Kanji mode, or condensed two-byte Kanji mode). In the single-byte ANK mode, they appear in single-byte code size; in the two-byte Kanji or condensed two-byte Kanji mode, they appear in half-width character size. (Note that the condensed two-byte Kanji mode is supported by the BHT-4000/BHT-5000.)

  In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, not only the screen mode but also the display font size determines the sizes of prompting message literals, echoed back literals, and cursor. If the standard-size font is selected, they appear in standard size; if the small-size font is selected, they appear in small size.

  In the BHT-7000/BHT-7500, in addition to the screen mode and display font size, the character width (normal-width or double-width) determines those sizes. If the double-width is selected, they appear in double width.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• Neither a comma (,) nor semicolon (;) follows `"`*`prompt`*`"`. • `"`*`prompt`*`"` is not a string constant.

Run-time errors:

Error code	Meaning
`06h`	The operation result is out of the allowable range. (Numeric *`variable`* is out of the range.)

Reference:

Statements:	`LINE INPUT` and `LOCATE`
Functions:	`INKEY$` and `INPUT$`

File I/O statement

INPUT

Reads data from a device I/O file into specified variables.

Syntax:

`INPUT #`*`filenumber`*`,`*`variable`*`[,`*`variable`*`...]`

Parameter:

filenumber

A numeric expression which returns a value from 1 to 16.

variable

A numeric or string variable.

Description:

`INPUT #` reads data from a device I/O file (a communications device file or bar code device file) specified by *filenumber* and assigns it to *variable*.

- *filenumber* is a number assigned to the device I/O file when it was opened.
- Reading data from a communications device file:

  An `INPUT #` statement reads data fields separated by CR codes or commas (,) and assigns them to *variable*.

  If more than one *variable* is specified in an `INPUT #` statement, the program waits until all of the specified *variable*s receive data.

  If an `INPUT #` statement reads data longer than the allowable string length, it ignores only the overflowed data and completes execution, causing no run-time error.
- Reading data from a bar code device file:

  An `INPUT #` statement reads the scanned data into the 1st *variable*.

  If more than one variable is specified in an `INPUT #` statement, the program ignores the 2nd and the following *variable*s.

  If an `INPUT #` statement reads data longer than the allowable string length, it ignores only the overflowed data and completes execution, causing no run-time error.

  In the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, if the maximum number of digits has been omitted in the read code specifications of the `OPEN "BAR:"` statement (except for the universal product codes), then the `INPUT #` statement can read bar codes of up to 99 digits. To read bar codes of 40 digits or more, you should define a sufficient string variable length beforehand.

- Notes for entering numeric data:

  Valid characters include 0 to 9, a minus sign (-), and a period (.). They should be in correct numeric data form. If not, `INPUT #` statement accepts only numeric data from the first character up to correctly formed character, as valid data. If no valid data is found, the `INPUT #` statement automatically assigns a zero (0) to *`variable`*.

  If the `INPUT #` statement reads alphabetical characters with a numeric variable, it automatically assigns a zero (0) to *`variable`*. For reading of Code 39 bar codes that may encode alphabetical characters, therefore, special care should be taken.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	*`filenumber`* is missing.

Run-time errors:

Error code	Meaning
`06h`	The operation result is out of the allowable range. (Numeric *`variable`* is out of the range.)
`34h`	Bad file name or number (You specified *`filenumber`* of an unopened file.)
`36h`	Improper file type (You specified *`filenumber`* of a file other than device I/O files.)
`3Ah`	File number out of the range

Example:

```
INPUT #fileNo,dat$
```

Reference:

Statements:	`CLOSE, LINE INPUT#, OPEN "BAR:"`, and `OPEN "COM:"`
Functions:	`INPUT$`

I/O statement

KEY

Assigns a string or a control code to a function key; also defines a function key as the LCD backlight function on/off key. This statement also defines a magic key as the trigger switch, shift key, or battery voltage display key.

Syntax:

Syntax 1 (Assigning a string or a control code to a function key):

KEY *keynumber*,*stringdata*

Syntax 2 (Defining a function key as the backlight function on/off key):

KEY *backlightkeynumber*,*onduration*

Syntax 3 (Defining a magic key as the battery voltage display key. Valid in the BHT-5000/BHT-6000/BHT-6500):

KEY *magickeynumber*,"BAT" (Battery voltage display key)

Syntax 4 (Defining a magic key as the trigger switch or shift key. Valid in the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500):

KEY *magickeynumber*,"TRG" (Trigger switch)
KEY *magickeynumber*,"SFT" (Shift key)

Parameter:

keynumber

(BHT-3000/BHT-4000)	A numeric expression which returns a value from 1 to 29.
(BHT-5000 with 32-key pad)	A numeric expression which returns a value from 1 to 46.
(BHT-5000 with 26-key pad)	A numeric expression which returns a value from 1 to 34.
(BHT-6000)	A numeric expression which returns a value from 1 to 31, 33, and 34.
(BHT-6500)	A numeric expression which returns a value from 1 to 31 and 33 to 38.
(BHT-7000 with 32-key pad/BHT-7500)	A numeric expression which returns a value from 1 to 31 and 33 to 50.
(BHT-7000 with 26-key pad)	A numeric expression which returns a value from 1 to 31 and 33 to 38.

stringdata

A string expression which returns up to two characters or a control code.

backlightkeynumber

(BHT-3000/BHT-4000)	A numeric expression which returns a value from 0 to 29.
(BHT-5000 with 32-key pad)	A numeric expression which returns a value from 0 to 46.
(BHT-5000 with 26-key pad)	A numeric expression which returns a value from 0 to 34.
(BHT-6000)	A numeric expression which returns a value from 1 to 31, 33, and 34.
(BHT-6500)	A numeric expression which returns a value from 1 to 31 and 33 to 38.
(BHT-7000 with 32-key pad/BHT-7500)	A numeric expression which returns a value from 1 to 31 and 33 to 50.
(BHT-7000 with 26-key pad)	A numeric expression which returns a value from 1 to 31 and 33 to 38.

onduration

Keyword BL and a string expression which returns a value from 0 to 255. (BL0 to BL255)

magickeynumber

(BHT-5000/BHT-6000)	30 or 31
(BHT-6500/BHT-7000 with 26-key pad)	30, 31, 35, or 36
(BHT-7000 with 32-key pad/BHT-7500)	30, 31, 47, 48

Description:

■ Assigning a string or a control code to a function key

KEY in syntax 1 assigns a string or a control code specified by *stringdata* to a function key specified by *keynumber*. Pressing the specified function key generates the assigned string data or control code and then passes it to the user program as if each character is keyed in directly from the keyboard.

- *keynumber* is a key number assigned to a particular function key. (Refer to Appendix E, "Key Number Assignment on the Keyboard.")
- In the BHT-5000, specifying 32 to *keynumber* assigns the trigger switch. In the BHT-6000/BHT-6500/BHT-7000 with 26-key pad/BHT-7500, specifying 32 will be ignored.
- *stringdata* is a character code ranging from 0 (00h) to 255 (FFh). (For the character codes, refer to Appendix C, "Character Sets.")
- If you specify more than two characters to *stringdata*, only the first two characters are valid.
- *stringdata* inputted by pressing the specified function key may be read to the user program by `INPUT` or `LINE INPUT` statement or `INKEY$` or `INPUT$` function.

  Note that `INKEY$` or `INPUT$ (1)` function can read only the first one character of the assigned two. The second character remains in the keyboard buffer and can be read by the `INPUT` or `LINE INPUT` statement or `INKEY$` or `INPUT$` function.

- If pressed together with the Shift key, any numerical key can operate as a function key.
- If you issue more than one KEY statement specifying a same function key, the last statement takes effect.
- If a null string is assigned to a function key, pressing the function key produces no key entry. To make a particular function key invalid, you specify a null string to *stringdata* as shown below.

```
KEY 1,""
KEY 2,CHR$(0)
KEY 3,CHR$(&h0)
```

■ Defining a function key as the LCD backlight function on/off key

KEY in syntax 2 defines a function key specified by *backlightkeynumber* as the backlight function on/off key and sets the length of backlight ON-time specified by *onduration*. (Refer to Chapter 13, "LCD Backlight Function.")

- *backlightkeynumber* is a key number assigned to a particular function key. (Refer to Appendix E, "Key Number Assignment on the Keyboard.")

  Pressing the specified backlight function on/off key activates or deactivates the backlight function.
- In the BHT-3000/BHT-4000/BHT-5000, specifying zero (0) to *backlightkey-number* restores the default (which is the combination of the trigger switch and shift key).
- In the BHT-6000/BHT-6500/BHT-7000 with 26-key pad/BHT-7500, specifying a zero (0) or 32 to *backlightkeynumber* will be ignored.
- In the BHT-3000/BHT-4000/BHT-5000, pressing the trigger switch while holding down the shift key functions as the backlight on/off control key by default. In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, pressing the M1 key (key number 33) while holding down the shift key functions as the backlight on/off control key by default.
- If pressed together with the Shift key, any numerical key can operate as a function key.
- *onduration* is the length of time in seconds from when the backlight is turned on to automatic turning-off. Pressing the trigger switch or any key (except for the backlight function on/off key) while the backlight is on resets the counter of *onduration* to the specified time length and restarts counting down.

  Specification of `BL0` disables the backlight function. Specification of `BL255` keeps the backlight on.
- A function key defined as the LCD backlight function on/off key cannot be used to enter string data.

- If you issue more than one KEY statement, the last statement takes effect. That is, if you define more than one key as the backlight function on/off key as shown below, only the function key numbered 8 operates as the backlight function on/off key and the length of backlight ON-time is 15 seconds.

```
KEY 5,"BL40"
KEY 8,"BL15"
```

■ Defining a magic key as the trigger switch, shift key, or battery voltage display key

- In the BHT-5000/BHT-6000/BHT-6500, KEY in syntax 3 defines a magic key as the trigger switch, shift key, or battery voltage display key as well as assigning string data.

```
KEY 30,"TRG"    M1 key as the trigger switch
KEY 31,"SFT"    M2 key as the shift key
KEY 30,"BAT"    SF+M1 keys as the voltage display key
```

- In the BHT-7000/BHT-7500, KEY in syntax 3 defines a magic key as the trigger switch or shift key as well as assigning string data. (It cannot define a magic key as the battery voltage display key.)

```
KEY 30,"TRG"    M1 key as the trigger switch
KEY 31,"SFT"    M2 key as the shift key
```

NOTE

If you issue KEY statements specifying a same function key, only the last KEY statement takes effect.

The description below, for example, makes the function key numbered 3 operate as the backlight function on/off key and the length of backlight ON-time is 100 seconds.

```
KEY 3,"a"
KEY 3,"BL100"
```

The description below assigns string data "a" to the function key numbered 3. The default backlight function on/off key (in the BHT-3000/BHT-4000/BHT-5000, the combination of the trigger switch and shift key; in the BHT-6000/BHT-6500/BHT-7000/BHT-7500, the combination of M1 key and shift key) will be restored.

```
KEY 3,"BL100"
KEY 3,"a"
```

The description below defines the magic key M1 as the trigger switch. The default battery voltage display key (combination of the ENT key and shift key) will be restored.

```
KEY 30,"BAT"
KEY 30,"TRG"
```

Syntax errors:

Error code and message	Meaning
`error 71:` `Syntax error`	• *`keynumber`* is missing. • *`stringdata`* is missing. • *`backlightkeynumber`* is missing. • *`stringdata`* is a numeric expression.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`keynumber`*, *`backlightkeynumber`*, or `magickeynumber` is out of the range.)

Example:

Syntax 1:

```
KEY 1,"a"
KEY 2,"F"+CHR$(13)
KEY 3,""
```

Syntax 2:

```
KEY 1,"BL60"
```

Reference:

Statements: `KEY OFF`, `KEY ON`, and `ON KEY`...`GOSUB`

KEY ON and KEY OFF

Enables or disables keystroke trapping for a specified function key.

Syntax:

```
KEY (keynumber){ON|OFF}
```

Parameter:

keynumber

(BHT-3000/BHT-4000)	A numeric expression which returns a value from 1 to 29.
(BHT-5000 with 32-key pad)	A numeric expression which returns a value from 1 to 46.
(BHT-5000 with 26-key pad)	A numeric expression which returns a value from 1 to 34.
(BHT-6000)	A numeric expression which returns a value from 1 to 31, 33, and 34.
(BHT-6500)	A numeric expression which returns a value from 1 to 31 and 33 to 38.
(BHT-7000 with 32-key pad/BHT-7500)	A numeric expression which returns a value from 1 to 31 and 33 to 50.
(BHT-7000 with 26-key pad)	A numeric expression which returns a value from 1 to 31 and 33 to 38.

Description:

■ KEY ON

KEY ON enables keystroke trapping for a function key specified by *keynumber*. (Refer to Appendix E, "Key Number Assignment on the Keyboard.")

- Between every execution of statements, the Interpreter checks whether a function key specified by the KEY ON statement is pressed or not. If the key is pressed, the Interpreter transfers control to the event-handling routine defined by an ON KEY...GOSUB statement before the KEY ON statement.
- If a function key which has been assigned a null string by the KEY statement is specified by the KEY ON statement, the keystroke trap takes place.
- If you specify a function key which has been defined as the LCD backlight function on/off key, trigger switch, shift key, or battery voltage display key by using the KEY ON statement, then no keystroke trap takes place.
- Keystroke trapping has priority over the INKEY$ function.

- When a program waits for the keyboard entry by the `INPUT`, `LINE INPUT` statement or `INPUT$` function, pressing a function key specified by the `KEY ON` statement neither reads the pressed key data nor causes keystroke trapping.
- In the BHT-6000/BHT-6500/BHT-7000 with 26-key pad/BHT-7500, specifying 32 to *`keynumber`* will be ignored.

■ `KEY OFF`

`KEY OFF` disables keystroke trapping for a function key specified by *`keynumber`*.

- In the BHT-6000/BHT-6500/BHT-7000 with 26-key pad/BHT-7500, specifying 32 to *`keynumber`* will be ignored.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• *`keynumber`* is not enclosed in parentheses ( ). • Neither `ON` or `OFF` follows (*`keynumber`*).

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`keynumber`* is out of the range.)

Reference:

Statements: `KEY` and `ON KEY...GOSUB`

KILL

Deletes a specified file from the memory.

Syntax:

```
KILL "[drivename:]filename"
```

Parameter:

`"[`*`drivename`*`:]`*`filename`*`"`

A string expression.

Description:

`KILL` deletes a data file or a user program file specified by `"[`*`drivename`*`:]`*`filename`*`"`.

In the BHT-5000/BHT-6000/BHT-6500, the *`drivename`* may be `A:` or `B:`. If the *`drivename`* is omitted, the default `A:` applies.

In the BHT-7000/BHT-7500, the *`drivename`* (`A:` or `B:`) will be ignored.

- The specified file will be deleted from both the data and the directory in the memory.
- A file to be deleted should be closed beforehand.

Syntax errors:

Error code and message	Meaning
`error 3: '"' missing`	No double quote precedes or follows `[`*`drivename`*`:]`*`filename`*`.`
`error 71: Syntax error`	`[`*`drivename`*`:]`*`filename`* is not enclosed in double quotes.

Run-time errors:

Error code	Meaning
`02h`	Syntax error (The format of `"[`*`drivename`*`:]`*`filename`*`"` is not correct.)
`35h`	File not found
`37h`	File already open

Example:

```
CLOSE
IF kyIn$="Y" THEN
  KILL "Master.Dat"
END IF
```

Reference:

Statements:	`CLFILE`

KPLOAD

Loads a user-defined Kanji font in the two-byte Kanji mode.
This statement also loads a user-defined cursor for the BHT-7000/BHT-7500.

Syntax:

Syntax 1 (Loading a user-defined Kanji font):

KPLOAD *kanjicode*, *fontarrayname*

Syntax 2 (Loading a user-defined cursor. Valid in the BHT-7000/BHT-7500):

KPLOAD *kanjicode*, *cursorarrayname*

Parameter:

kanjicode

- For a user-defined Kanji font

(BHT-3000/BHT-4000/BHT-5000/BHT-6000/BHT-6500)	A numeric expression which returns a value from EBC0h to EBDFh.
(BHT-7000/BHT-7500)	A numeric expression which returns a value from EBC0h to EBDFh, EC40h to EC7Eh, and EC80h to EC83h.

- For a user-defined cursor

  A numeric expression which returns zero (0).

fontarrayname and *cursorarrayname*

An array integer variable name.

NOTE Do not specify parentheses ( ) or subscripts which represent a general array as shown below. It will result in an error.

```
KPLOAD &HEBC0,kp%()  'error
KPLOAD &HEBC0,kp%(2) 'error
```

Description:

■ Loading a user-defined Kanji font

KPLOAD loads a user-defined Kanji font data defined by *fontarrayname* to the user font area specified by *kanjicode*.

- *kanjicode* is a shift JIS code.
- To display user-defined Kanji fonts loaded by the KPLOAD, you use the PRINT statement in the two-byte Kanji mode. If you attempt to display an undefined Kanji character code, a full-width space character will appear.

- The loaded user-defined fonts are effective during execution of the user program which loaded those fonts and during execution of the successive user programs chained by the `CHAIN` statement.
- If you load a font to the same *`kanjicode`* more than one time, the most recently specified font takes effect.
- Only when the Interpreter executes the `KPLOAD` statement, it refers to the array data defined by *`fontarrayname`*. So, once a user program has finished loading the user font, changing the data in the array or deleting the array itself (by the `ERASE` statement) will not affect the already loaded user font.
- An array integer variable--a work array, register array, or common array--for *`fontarrayname`* should be declared by the `DIM`, `DEFREG`, or `COMMON` statement, respectively.

  ```
  DIM kp0%(15)
  DEFREG kp1%(15)
  COMMON kp2%(15)
  ```

  The array variable should be one-dimensional and have at least 16 elements. Each element data should be an integer and stored in the area from the 1st to 16th elements of the array.
- The loaded user-defined fonts are valid also in the condensed two-byte Kanji mode (BHT-4000/BHT-5000). They are also effective when the small-size font is selected (BHT-6000/BHT-6500/BHT-7000/BHT-7500). Note that the dot pattern of each character will be condensed by the system program. (For the generating procedure of condensed user-defined fonts, refer to Appendix C3., "Display Mode and Letter Size.")
- The loaded user-defined fonts are valid also in the double-width mode (BHT-7000/BHT-7500). Note that the dot pattern of each character will be doubled in width by the system program.

■ Loading a user-defined cursor (BHT-7000/BHT-7500)

`KPLOAD` loads a user-defined cursor data defined by *`cursorarrayname`* to the user font area specified by *`kanjicode`*.

- To display user-defined cursors loaded by the `KPLOAD`, you use the `LOCATE` statement in the two-byte Kanji mode, in which you set 255 to *`cursorswitch`* (`LOCATE ,,255`).
- The loaded user-defined cursors are effective during execution of the user program which loaded those cursors and during execution of the successive user program chained by the `CHAIN` statement.
- Only when the Interpreter executes the `KPLOAD` statement, it refers to the array data defined by *`cursorarrayname`*. So, once a user program has finished loading the user cursor, changing the data in the array or deleting the array itself (by the ERASE statement) will not affect the already loaded user cursor.

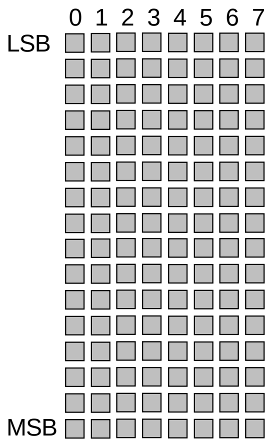
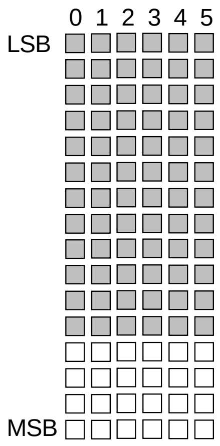
- An array integer variable--a work array, register array, or common array--for *`cursorarrayname`* should be declared by the `DIM`, `DEFREG`, or `COMMON` statement, respectively.

```
DIM kp0%(5)
DEFREG kp1%(5)
COMMON kp2%(5)
```

  The array variable should be one-dimensional and have at least 6 elements. Each element data should be an integer and stored in the area from the 1st to 6th elements of the array.

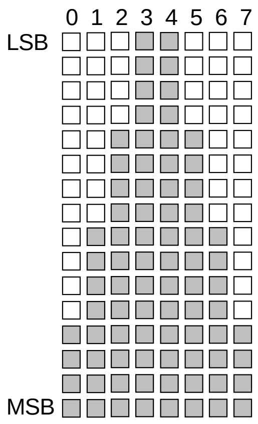
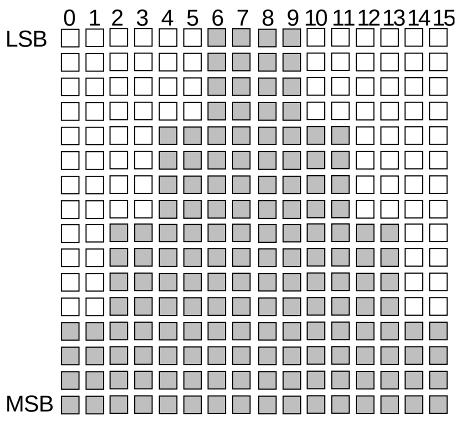
- If the cursor size (the number of elements in an array variable wide by the number of bits high) defined by *`cursorarrayname`* exceeds the allowable size, the excess will be discarded.

- The cursor size will be as follows depending upon the font size.

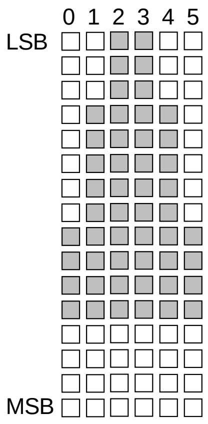
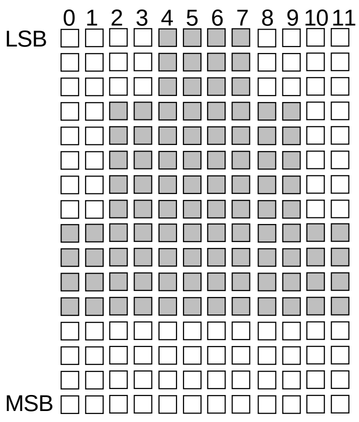
Font size	Cursor size (W x H)	No. of elements
Standard-size	8 x 16 dots 	8
Small-size	6 x 12 dots 	6

- In double-width display mode, the cursor will appear in double width as shown below:

  When the standard-size font is selected

Cursor loaded	Cursor displayed in double width
	

  When the small-size font is selected

Cursor loaded	Cursor displayed in double width
	

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• No *`fontarrayname`* or *`cursor-arrayname`* is defined. • *`fontarrayname`* or *`cursorar-rayname`* has an array string vari-able. • *`fontarrayname`* or *`cursorar-rayname`* includes parentheses ( ). • *`fontarrayname`* or *`cursorar-rayname`* includes subscripts.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (• *`kanjicode`* is out of the range.) (• *`fontarrayname`* or *`cursorarrayname`* is not correct.)
`08h`	Array not defined

Example:

```
DIM kp%(15)
kp%(0)=&H0000
kp%(1)=&H8011
kp%(2)=&H6022
kp%(3)=&H1844
kp%(4)=&H0600
kp%(5)=&H8802
kp%(6)=&H8AF2
kp%(7)=&H4A92
kp%(8)=&H4A97
kp%(9)=&H2A92
kp%(10)=&H1FF2
kp%(11)=&H2A92
kp%(12)=&H4A97
kp%(13)=&H4A92
kp%(14)=&H8AF2
kp%(15)=&H8802
   ⋮
SCREEN 1
KPLOAD &HEBC0,kp%
PRINT CHR$(&HEB);CHR$(&HC0)
```

Array Elements

kp%(0)					kp%(5)					kp%(10)					kp%(15)	Bit in each array element
□	■	□	□	□	□	□	□	■	□	□	□	■	□	□	□	0 (LSB)
□	□	■	□	□	■	■	■	■	■	■	■	■	■	■	■	1
□	□	□	■	□	□	□	□	■	□	□	□	■	□	□	□	2
□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	3
□	■	□	□	□	□	■	■	■	■	■	■	■	■	■	□	4
□	□	■	□	□	□	■	□	□	□	■	□	□	□	■	□	5
□	□	□	■	□	□	■	□	□	□	■	□	□	□	■	□	6
□	□	□	□	□	□	■	■	■	■	■	■	■	■	■	□	7
□	□	□	□	□	□	□	□	□	□	■	□	□	□	□	□	8
□	□	□	□	■	□	■	■	■	■	■	■	■	■	■	□	9
□	□	□	□	■	□	□	□	□	□	■	□	□	□	□	□	10
□	□	□	■	□	□	□	□	□	□	■	□	□	□	□	□	11
□	□	□	■	□	■	■	■	■	■	■	■	■	■	■	■	12
□	□	■	□	□	□	□	□	□	■	□	■	□	□	□	□	13
□	□	■	□	□	□	□	■	■	□	□	□	■	■	□	□	14
□	■	□	□	□	■	■	□	□	□	□	□	□	□	■	■	15 (MSB)

Reference:

Statements: `APLOAD`, `COMMON`, `DEFREG`, `DIM`, `PRINT`, and `SCREEN`

Assignment statement

LET

Assigns a value to a given variable.

Syntax:

Syntax 1:

`[LET]` *`stringvariable = stringexpression`*

Syntax 2:

`[LET]` *`numericvariable = numericexpression`*

Description:

LET assigns a value of expression on the right-hand side to a variable on the left-hand side.

- In a numeric data assignment, the assignment statement automatically converts an integer value to a real value. In the type conversion from a real value to an integer value, it rounds off the fractional part.
- Keyword LET can be omitted since the equal sign is all that is required to assign a value.
- The data type of a variable and an expression must correspond.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	The data type on the right- and left-hand sides does not correspond. That is, the variable on the left-hand side is numeric but the expression on the right-hand side is a string, or vice versa.

Run-time errors:

Error code	Meaning
`06h`	The operation result is out of the allowable range.
`0Fh`	String length out of the range (In a string assignment, the string length of the evaluated result on the right-hand side exceeds the maximum length of the string variable on the left-hand side.)
`10h`	Expression too long or complex

I/O statement

LINE INPUT

Reads input from the keyboard into a string variable.

Syntax:

`LINE INPUT ["`*`prompt`*`"{,|;}]`*`stringvariable`*

Parameter:

"*prompt*"

A string constant.

stringvariable

A string variable.

Description:

When execution reaches a `LINE INPUT` statement, the program pauses and waits for the user to enter data from the keyboard while showing a prompting message specified by "*prompt*".

After typing data, the user must press the ENT key. Then, the `LINE INPUT` statement assigns the typed data to *stringvariable*.

- A `LINE INPUT` statement cannot assign a numeric variable. (An `INPUT` statement can do.)
- "*prompt*" is a prompting message to be displayed on the LCD.
- The semicolon (;) or comma (,) after "*prompt*" has the following meaning:

  If "*prompt*" is followed by a semicolon, the `LINE INPUT` statement displays the prompting message followed by a question mark and a space.

```
LINE INPUT "data= ";a$
```

```
data= ?
```

If "*prompt*" is followed by a comma, the statement displays the prompting message but no question mark or space is appended to the prompting message.

```
LINE INPUT "data= ",a$
```

```
data=
```

- The cursor shape specified by the most recently executed `LOCATE` statement takes effect.
- Even after execution of the `CURSOR OFF` statement, the `LINE INPUT` statement displays the cursor.
- Data inputted by the user will echo back to the LCD. To assign it to *stringvariable*, it is necessary to press the ENT key.

  Pressing the ENT key causes also a line feed.

  If you type no data and press the ENT key, a `LINE INPUT` statement automatically assigns a null string to *stringvariable*.
- When any echoed back data is displayed on the LCD, pressing the Clear or BS key erases the whole displayed data or a most recently typed-in character of the data, respectively. If no data is displayed, pressing the Clear or BS key produces no operation.
- Notes for entering string data:

  The effective length of string data is the maximum string length of *stringvariable*. Overflowed data will be ignored.
- The sizes of prompting message literals, echoed back literals and cursor depend upon the screen mode (the single-byte ANK mode, two-byte Kanji mode, or condensed two-byte Kanji mode). In the single-byte ANK mode, they appear in single-byte code size; in the two-byte Kanji or condensed two-byte Kanji mode, they appear in half-width character size. (Note that the condensed two-byte Kanji mode is supported by the BHT-4000/BHT-5000.)

  In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, not only the screen mode but also the display font size determines the sizes of prompting message literals, echoed back literals, and cursor. If the standard-size font is selected, they appear in standard size; if the small-size font is selected, they appear in small size.

  In the BHT-7000/BHT-7500, in addition to the screen mode and display font size, the character width (normal-width or double-width) determines those sizes. If the double-width is selected, they appear in double width.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• `INPUT` is missing. • Neither a comma (,) or semicolon (;) follows `"`*`prompt`*`"`. • `"`*`prompt`*`"` is not a string constant. • *`stringvariable`* has a numeric variable. • A semicolon (;) immediately follows `LINE INPUT`.

Reference:

Statements:	`INPUT` and `LOCATE`
Functions:	`INKEY$` and `INPUT$`

LINE INPUT

Reads data from a device I/O file into a string variable.

Syntax:

`LINE INPUT #`*`filenumber`*`,`*`stringvariable`*

Parameter:

`filenumber`

A numeric expression which returns a value from 1 to 16.

`stringvariable`

A string variable.

Description:

`LINE INPUT #` reads data from a device I/O file (a communications device file or bar code device file) specified by *`filenumber`* and assigns it to *`stringvariable`*.

- *`filenumber`* is a number assigned to the device I/O file when it was opened.
- A `LINE INPUT #` statement cannot assign a numeric variable. (An `INPUT #` statement can do.)
- Reading data from a communications device file:

  A `LINE INPUT #` statement reads all of the string literals preceding a CR code and assigns them to *`stringvariable`* except for CR codes and LF codes which immediately follow a CR code.

  If a `LINE INPUT #` statement reads data longer than the allowable string length before reading a CR code, it ignores only the overflowed data and completes execution, causing no run-time error.
- Reading data from a bar code device file:

  A `LINE INPUT #` statement reads the scanned data into *`stringvariable`*.

  If a `LINE INPUT #` statement reads data longer than the allowable string length, it ignores only the overflowed data and completes execution, causing no run-time error.

  In the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, if the maximum number of digits has been omitted in the read code specifications of the `OPEN "BAR:"` statement (except for the universal product codes), then the `INPUT #` statement can read bar codes of up to 99 digits. To read bar codes exceeding 40 digits, you should define a sufficient string variable length beforehand.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• `INPUT` is missing. • *`filenumber`* is missing. • `"`*`prompt`*`"` is not a string constant. • *`stringvariable`* has a numeric variable.

Run-time errors:

Error code	Meaning
`34h`	Bad file name or number (You specified *`filenumber`* of an unopened file.)
`36h`	Improper file type (You specified *`filenumber`* of a file other than device I/O files.)
`3Ah`	File number out of the range

Example:

```
LINE INPUT #fileNo,dat$
```

Reference:

Statements:	`CLOSE`, `INPUT#`, `OPEN "BAR:"`, and `OPEN "COM:"`
Functions:	`INPUT$`

LOCATE

Moves the cursor to a specified position and changes the cursor shape.

Syntax:

Syntax 1:

```
LOCATE [column][,row[,cursorswitch]]
```

Syntax 2:

```
LOCATE,,cursorswitch
```

Parameter:

A numeric expression which returns a value given below.

Single-byte ANK Mode

	BHT-3000	BHT-4000	BHT-5000	BHT-6000/BHT-6500	
				Standard-size font	Small-size font
column	1 to 17	1 to 27	1 to 22	1 to 17	1 to 17
row	1 to 4	1 to 10 (1 to 9*)	1 to 8	1 to 6	1 to 8
cursorswitch	0 to 2	0 to 2	0 to 2	0 to 2	0 to 2

	BHT-7000		BHT-7500	
	Standard-size font	Small-size font	Standard-size font	Small-size font
column	1 to 22	1 to 22	1 to 27	1 to 27
row	1 to 8	1 to 10	1 to 20	1 to 26
cursorswitch	0 to 2, 255	0 to 2, 255	0 to 2, 255	0 to 2, 255

* Values in parentheses will be returned when the system status indication is set to ON. If you specify the bottom line of the LCD as the desired cursor position when the system status is displayed, the cursor cannot move to the bottom line and it will move to the next to the bottom line instead.

Two-byte Kanji Mode

	BHT-3000	BHT-4000	BHT-5000	BHT-6000/BHT-6500 Standard-size font	BHT-6000/BHT-6500 Small-size font
column	1 to 13	1 to 21	1 to 17	1 to 13	1 to 17
row	1 to 3	1 to 9 (1 to 8*)	1 to 7	1 to 5	1 to 7
cursorswitch	0 to 2	0 to 2	0 to 2	0 to 2	0 to 2

	BHT-7000		BHT-7500	
	Standard-size font	Small-size font	Standard-size font	Small-size font
column	1 to 17	1 to 22	1 to 21	1 to 27
row	1 to 7	1 to 9	1 to 19	1 to 25
cursorswitch	0 to 2, 255	0 to 2, 255	0 to 2, 255	0 to 2, 255

Condensed Two-byte Kanji Mode

	BHT-4000	BHT-5000
column	1 to 27	1 to 22
row	1 to 9 (1 to 8*)	1 to 7
cursorswitch	0 to 2	0 to 2

* Values in parentheses will be returned when the system status indication is set to ON. If you specify the bottom line of the LCD as the desired cursor position when the system status is displayed, the cursor cannot move to the bottom line and it will move to the next to the bottom line instead.

Description:

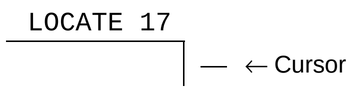
LOCATE moves the cursor to a position specified by *column* number and *row* number as co-ordinates on the LCD. It also changes the cursor shape as specified by *cursorswitch*.

- The cursor location in the upper left corner of the LCD is 1, 1 which is the default.
- *cursorswitch* specifies the cursor shape as listed below.

cursorswitch value	Cursor shape
0	Invisible
1	Underline cursor (default)
2	Full block cursor
255	User-defined cursor (valid in the BHT-7000/ BHT-7500 only)

- Specification of the maximum value to *`column`* moves the cursor off the screen and out of sight.

  Example: Single-byte ANK mode in the BHT-3000



  If you display data on the screen under the above condition, the cursor moves to the 1st column of the next row, from where the data appears.

- In the BHT-4000, if the system status indication is set to ON, the cursor cannot move to the bottom line of the LCD. If you specify the bottom line, the cursor will move to the next to the bottom line instead.
- In the BHT-5000/BHT-6000/BHT-6500, if you specify the right end of the bottom line as the desired cursor position when the system status is displayed, the cursor becomes invisible.
- If some parameter is omitted, the current value remains active. If you omit *`column`*, for example, the cursor stays in the same column but moves to the newly specified row position.
- Any parameter value outside the range will result in a run-time error.
- The column range does not differ between the normal- and double-width characters.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range

Example:

```
LOCATE 1,2
LOCATE xPos,xCSRLIN
LOCATE ,,2
```

Reference:

Functions:	`CSRLIN` and `POS`

Error control statement

ON ERROR GOTO

Enables error trapping.

Syntax:

```
ON ERROR GOTO label
```

Description:

`ON ERROR GOTO` enables error trapping so as to pass control to the first line of an error-handling routine specified by *`label`* if an error occurs during program execution.

- To return control from an error-handling routine to a specified program location, you use a `RESUME` statement in the error-handling routine.
- Specification of zero (`0`) to *`label`* disables error trapping.

  If `ON ERROR GOTO 0` is executed outside the error-handling routine, the occurrence of any subsequent error displays a regular run-time error code and terminates the program.

  If `ON ERROR GOTO 0` is executed inside the error-handling routine, the Interpreter immediately displays the regular run-time error code and terminates the program.
- You cannot trap errors which may occur during execution of the error-handling routine. The occurrence of such an error displays a run-time error code and terminates the program.
- You can use `ON ERROR GO TO` instead of `ON ERROR GOTO`.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• *`label`* has not been defined. • *`label`* is missing.

Reference:

Statements:	`RESUME`
Functions:	`ERL` and `ERR`

ON...GOSUB and ON...GOTO

Branches to one of specified labels according to the value of an expression.

Syntax:

Syntax 1:

`ON` *`expression`* `GOSUB` *`label`* `[,`*`label`*`...]`

Syntax 2:

`ON` *`expression`* `GOTO` *`label`* `[,`*`label`*`...]`

Parameter:

`expression`

A numeric expression which returns a value from 1 to 255.

Description:

`ON`...`GOSUB` or `ON`...`GOTO` block branches to a *`label`* in the label list according to the value of *`expression`*.

- If *`expression`* has the value 3, for example, the target label is the third one in the label list counting from the first.
- If *`expression`* has the value 0 or a value greater than the number of labels in the label list, execution of the `ON`...`GOSUB` or `ON`...`GOTO` block causes no run-time error and passes control to the subsequent statement.
- You can specify any number of labels so long as a statement block does not exceed one program line (512 characters).
- You can nest `ON`...`GOSUB` statements to a maximum of 10 levels.
- When using the `GOSUB` statement together with block-structured statements (`DEF FN`...`END DEF`, `FOR`...`NEXT`, `FUNCTION`...`END FUNCTION`, `IF`...`THEN` ...`ELSE`...`END IF`, `SELECT`...`CASE`...`END SELECT`, `SUB`...`END SUB` and `WHILE`...`WEND`), you can nest them to a maximum of 30 levels.
- You can use `ON`...`GO TO` instead of `ON`...`GOTO`.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• *`label`* has not been defined. • *`label`* is missing.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`expression`* is negative or greater than 255.)
`07h`	Insufficient memory space (The program nesting by `GOSUB` statements only is too deep.)

Reference:

Statements:	`GOSUB`, `GOTO`, and `SELECT...CASE...END SELECT`

ON KEY...GOSUB

Specifies an event-handling routine for keystroke interrupt.

Syntax:

```
ON KEY (keynumber) GOSUB label
```

Parameter:

keynumber

(BHT-3000/BHT-4000)	A numeric expression which returns a value from 1 to 29.
(BHT-5000 with 32-key pad)	A numeric expression which returns a value from 1 to 46.
(BHT-5000 with 26-key pad)	A numeric expression which returns a value from 1 to 34.
(BHT-6000)	A numeric expression which returns a value from 1 to 31, 33, and 34.
(BHT-6500)	A numeric expression which returns a value from 1 to 31 and 33 to 38.
(BHT-7000 with 32-key pad/BHT-7500)	A numeric expression which returns a value from 1 to 31 and 33 to 50.
(BHT-7000 with 26-key pad)	A numeric expression which returns a value from 1 to 31 and 33 to 38.

Description:

According to *label*, ON KEY...GOSUB specifies the first line of an event-handling routine to be invoked if a function key specified by *keynumber* is pressed. (Refer to Appendix E, "Key Number Assignment on the Keyboard.")

- ON KEY...GOSUB specifies only the location of an event-handling routine but does not enable keystroke trapping. It is KEY ON statement that enables keystroke trapping. (Refer to KEY ON and KEY OFF.)
- Specification of zero (0) to *label* disables keystroke trapping.

- If a keystroke trap occurs, the Interpreter automatically executes `KEY OFF` statement for the pressed function key before passing control to an event-handling routine specified by *`label`* in `ON KEY...GOSUB` statement. This prevents a same event-handling routine from becoming invoked again by pressing a same function key during execution of the routine until the current event-handling routine is completed by issuing a `RETURN` statement.

  When control returns from the event-handling routine by a `RETURN` statement, the Interpreter automatically executes `KEY ON` statement.

  If it is not necessary to resume keystroke trapping, you describe a `KEY OFF` statement in the event-handling routine.
- If you issue more than one `ON KEY...GOSUB` statement specifying a same *`keynumber`*, the last statement takes effect.
- You can nest `GOSUB` statements to a maximum of 10 levels.
- When using the `ON KEY...GOSUB` statement together with block-structured statements (`DEF FN...END DEF`, `FOR...NEXT`, `FUNCTION...END FUNCTION`, `IF...THEN...ELSE...END IF`, `SELECT...CASE...END SELECT`, `SUB...END SUB` and `WHILE...WEND`), you can nest them to a maximum of 30 levels.
- In the BHT-6000/BHT-6500/BHT-7000 with 26-key pad/BHT-7500, specifying 32 to *`keynumber`* will be ignored.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• *`label`* has not been defined. • *`label`* is missing. • *`keynumber`* is not enclosed in parentheses ( ).

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`keynumber`* is out of the range.)
`07h`	Insufficient memory space (The program nesting by `GOSUB` statements is too deep.)

Reference:

Statements: `KEY`, `KEY OFF`, and `KEY ON`

OPEN

Opens a file for I/O activities.

Syntax:

`OPEN "[`*`drivename`*`:]`*`filename`*`" AS [#]` *`filenumber`* `[RECORD` *`filelength`*`]`

Parameter:

`filenumber`

A numeric expression which returns a value from 1 to 16.

`"[`*`drivename`*`:]`*`filename`*`"`

A string expression.

`filelength`

An integer constant which has the value from 1 to 32767.

Description:

`OPEN` opens a data file specified by `"[`*`drivename`*`:]`*`filename`*`"` and associates the opened file with *`filenumber`* for allowing I/O activities according to *`filenumber`*.

- The maximum number of files which can be opened at one time is 16 including the bar code device file and communications device files.
- `"`*`filename`*`"` consists of a file name and a file extension.

  The file name should be 1 to 8 characters long. Usable characters for the file name include alphabet letters, numerals, a minus (-) sign, and an underline (_). Note that a minus sign and underline should not be used for the starting character of the file name. Uppercase and lowercase alphabet letters are not distinguished from each other and both are treated as uppercase letters.

  The file extension should be up to 3 characters long. It should be other than .PD3, .EX3, .FN3, and .FLD and may be omitted (together with a period).

  ```
  a.dat
  master01.dat
  ```

- For the BHT-5000/BHT-6000/BHT-6500, the *`drivename`* may be `A:` or `B:`. If the *`drivename`* is omitted, the default `A:` applies.

- In the BHT-7000/BHT-7500, if the `drivename` is `B:`, the file specified by `filename` will be opened as a read-only file. If the `drivename` is `A:` or omitted, the file will be opened as a read/write file.
- `filelength` is the maximum number of registrable records in a file. It can be set only when a new data file is created by an `OPEN` statement. If you specify `filelength` when opening any of existing data files (including downloaded data files), the `filelength` will be ignored.
- Specifying `filelength` does not allocate memory. Therefore, whether or not a `PUT` statement can write records up to the specified `filelength` depends on the memory occupation state.
- If `filelength` is omitted, the default file size is 1000 records.

Syntax errors:

Error code and message	Meaning
`error 3: '"' missing`	No double quote precedes or follows `"[drivename:]filename"`.
`error 71: Syntax error`	• `filelength` is out of the range. • `filelength` is not an integer constant. • `"[drivename:]filename"` is not enclosed in double quotes.

Run-time errors:

Error code	Meaning
02h	Syntax error (`"[drivename:]filename"` is not correct. Or the bar code device file or communications device file is specified.)
07h	Insufficient memory space
32h	File type mismatch
37h	File already open
3Ah	File number out of the range
41h	File damaged

Reference:

Statements: `CLOSE`, `OPEN "BAR:"`, and `OPEN "COM:"`

OPEN "BAR:"

Opens the bar code device file. In the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, this statement also activates or deactivates the reading confirmation LED and the beeper (vibrator) individually. (Vibrator control valid only in the BHT-6500/BHT-7000/BHT-7500)

Syntax:

OPEN "BAR:[*readmode*][*beepercontrol*][*LEDcontrol*]" AS [#]*filenumber* CODE *readcode*[,*readcode*...]

Parameter:

readmode

A string expression.

beepercontrol *(for the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500)*

A string expression. Specification of B activates the beeper (vibrator). (Default: Deactivated)

LEDcontrol *(for the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500)*

A string expression. Specification of L deactivates the reading confirmation LED. (Default: Activated)

filenumber

A numeric expression which returns a value from 1 to 16.

readcode

A string expression.

Description:

OPEN "BAR:" opens the bar code device file and associates it with *filenumber* for allowing data entry from the bar code reader according to *filenumber*.

If the bar code device file has been opened with the OPEN "BAR:" statement, pressing the trigger switch in the BHT-3000 makes the illumination LED start blinking; pressing the trigger switch[*1] in the BHT-4000/BHT-5000/BHT-6000/BHT-7000 turns on the illumination LED; pressing the trigger switch[*1] in the BHT-6500/BHT-7500 emits a laser beam[*2].

In the BHT-3000, when you bring the BHT near bar codes, the illumination LED comes to stay on.

[*1] In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, the trigger switch function is assigned to the magic keys.

[*2] The BHT-6500/BHT-7500 uses a laser source.

- If the BHT reads a bar code successfully, the indicator LED for reading confirmation will illuminate in green. The BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500 may activate or deactivate the indicator LED. The BHT-6500/BHT-7000/BHT-7500 may activate or deactivate the beeper and vibrator function.
- A bar code read will be decoded and then transferred to the barcode buffer. In the BHT-3000, if the decoded data does not satisfy the reading conditions*, then the reading confirmation LED will illuminate in red and no data will be transferred to the barcode buffer.

  (*The reading conditions include the number of digits, a check digit, the type of the leading character, and start/stop characters.)
- Only a single bar code device file can be opened at a time. The total number of files which can be opened at a time is 16 including data files and communications device files.
- The BHT-6000/BHT-6500/BHT-7000/BHT-7500 cannot open the bar code device file and the optical interface of the communications device file concurrently. If you attempt to open them concurrently, a run-time error will occur. The BHT can open the bar code device file and the direct-connect interface concurrently.
- The name of the bar code device file, BAR, may be in lowercase.

```
OPEN "bar:" AS #10 CODE "A"
```

- Alphabet letters to be used for *readmode*, *beepercontrol*, *LEDcontrol* and *readcode* may be in lowercase.
- Up to eight *readcode*s can be specified.
- If you specify more than one condition to a same read code, all of them are valid. The sample below makes the BHT read both of the 6- and 10-digit ITF codes.

```
OPEN "BAR:" AS #1 CODE "I:6","I:10"
OPEN "BAR:" AS #1 CODE "I:6,10"
```
(For the BHT-6500/BHT-7000/BHT-7500)

■ readmode

The BHT supports four read modes--the momentary switching mode, the auto-off mode, the alternate switching mode, and the continuous reading mode, which can be selected by specifying M, F, A, and C to *readmode*, respectively.

□Momentary switching mode (M)

```
OPEN "BAR:M" AS #7 CODE "A"
```

Only while you hold down the trigger switch[*1], the illumination LED (laser source[*2]) lights and the BHT can read a bar code.

In the BHT-3000/BHT-4000/BHT-5000/BHT-6000, even if the bar code device file is closed, the illumination LED does not go off so long as the trigger switch[*1] is held down.

In the BHT-6500/BHT-7000/BHT-7500, if the bar code device file is closed when the trigger switch[*1] is held down, the illumination LED (laser source[*2]) will go off.

Until the entered bar code data is read out from the barcode buffer, pressing the trigger switch[*1] cannot turn on the illumination LED (laser source[*2]) so that the BHT cannot read the next bar code.

[*1] In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, the trigger switch function is assigned to the magic keys.

[*2] The BHT-6500/BHT-7500 uses a laser source.

☐Auto-off mode (F)

```
OPEN "BAR:F" AS #7 CODE "A"
```

If you press the trigger switch[*1], the illumination LED (laser source[*2]) comes on. When you release the switch or when the BHT completes bard code reading, the illumination LED (laser source[*2]) will go off. Holding down the trigger switch[*1] lights the illumination LED (laser source[*2]) for a maximum of 5 seconds.

While the illumination LED (laser source[*2]) is on, the BHT can read a bar code until a bar code is read successfully or the bar code devices file becomes closed.

If the illumination LED (laser source[*2]) goes off after 5 seconds from when you press the trigger switch[*1], it is necessary to press the trigger switch[*1] again for reading a bar code.

In the BHT-3000/BHT-4000/BHT-5000/BHT-6000, once a bar code is read successfully, pressing the trigger switch[*1] turns on the illumination LED (laser source[*2]) but the BHT cannot read the next bar code as long as the entered bar code data is not read out from the barcode buffer.

In the BHT-6500/BHT-7000/BHT-7500, once a bar code is read successfully, pressing the trigger switch[*1] cannot turn on the illumination LED (laser source[*2]) and the BHT cannot read the next bar code as long as the entered bar code data is not read out from the barcode buffer.

☐Alternate switching mode (A)

```
OPEN "BAR:A" AS #7 CODE "A"
```

If you press the trigger switch[*1], the illumination LED (laser source[*2]) comes on. Even if you release the switch, the illumination LED (laser source[*2]) remains on until the bar code device file becomes closed or you press that switch again. While the illumination LED (laser source[*2]) is on, the BHT can read a bar code.

Pressing the trigger switch[*1] toggles the illumination LED (laser source[*2]) on and off.

Once a bar code is read successfully, pressing the trigger switch[*1] turns on the illumination LED (laser source[*2]) but the BHT cannot read the next bar code as long as the entered bar code data is not read out from the barcode buffer.

[*1] In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, the trigger switch function is assigned to the magic keys.

[*2] The BHT-6500/BHT-7500 uses a laser source.

□**Continuous reading mode (`C`)**

```
OPEN "BAR:C" AS #7 CODE "A"
```

Upon execution of the above statement, the BHT turns on the illumination LED (laser source[*2]) and keeps it on until the bar code device file becomes closed, irrespective of the trigger switch[*1].

While the illumination LED (laser source[*2]) is on, the BHT can read a bar code.

Once a bar code is read successfully, the BHT cannot read the next bar code as long as the entered bar code data is not read out from the barcode buffer.

- If *`readmode`* is omitted, the BHT defaults to the auto-off mode.
- In the momentary switching mode, alternate switching mode, or continuous reading mode, after you read a low-quality bar code which needs more than one second to be read, keeping applying the barcode reading window to that bar code may re-read the same bar code in succession at intervals of one second or more.

■ **`beepercontrol`** and **`LEDcontrol`** (for the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500)

In the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, the `OPEN "BAR:"` statement can control the beeper and the reading confirmation LED to activate or deactivate each of them when a bar code is read successfully. The BHT-6500/BHT-7000/BHT-7500 may also control the vibrator with *`beepercontrol`*.

- You should describe parameters of *`readmode`*, *`beepercontrol`*, and *`LEDcontrol`* without any space inbetween.
- You should describe *`readmode`*, *`beepercontrol`*, and *`LEDcontrol`* in this order.
- In the BHT-6500/BHT-7000/BHT-7500, specifying `B` to *`beepercontrol`* allows you to choose beeping only, vibrating only, or beeping & vibrating by making setting on the "LCD contrast & beeper volume screen" or by setting the I/O ports with the `OUT` statement.

  To sound the beeper when a bar code is read successfully:

  ```
  OPEN "BAR:B" AS #7 CODE "A"
  ```

  To deactivate the reading confirmation LED when a bar code is read successfully:

  ```
  OPEN "BAR:L" AS #7 CODE "A"
  ```

[*1] In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, the trigger switch function is assigned to the magic keys.

[*2] The BHT-6500/BHT-7500 uses a laser source.

■ readcode

The BHT supports six types of bar codes--the universal product codes, Interleaved 2 of 5 (ITF), Codabar (NW-7), Code 39, Code 93, and Code 128. In addition to them, the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500 supports the Standard 2 of 5 (STF). The BHT-6000/BHT-6500/BHT-7000/BHT-7500 can read also EAN-128 if Code 128 is specified.

(For the allowable bar code types, refer to the BHT User's Manual.)

□Universal product codes (A)

Syntax 1:

```
A[:[code][1stchara[2ndchara]][supplemental]]
```

Syntax 2 (BHT-6500/BHT-7000/BHT-7500):

```
A[:[code][1stchara[2ndchara]][supplemental]
 [,[code][1stchara[2ndchara]][supplemental]]
 [,[code][1stchara[2ndchara]][supplemental]]]
```

where

code is A, B, or C specifying the following:

code	Bar code
A	EAN-13 or UPC-A
B	EAN-8
C	UPC-E

If *code* is omitted, the default is all of the universal product codes.

1stchara or *2ndchara* is a numeral from 0 to 9 specifying the header character (country flag). If a question mark (?) is specified to *1stchara* or *2ndchara*, it acts as a wild card.

supplemental is a supplemental code. Specifying an S to *supplemental* allows the BHT (expect for the BHT-3000) to read also supplemental codes. The BHT-3000 does not support supplemental codes, so specifying the *supplemental* option will cause a run-time error.

```
OPEN "BAR:" AS #1 CODE "A:49S"
```

□Interleaved 2 of 5 (ITF) (`I`)

Syntax 1:

```
I[:[mini.no.digits[-max.no.digits]][CD]]
```

Syntax 2 (BHT-6500/BHT-7000/BHT-7500):

```
I[:[mini.no.digits[-max.no.digits]][CD]
 [,[mini.no.digits[-max.no.digits]][CD]]
 [,[mini.no.digits[-max.no.digits]][CD]]]
```

where

mini.no.digits and *max.no.digits* are the minimum and maximum numbers of digits for bar codes to be read by the BHT, respectively.

They should be a numeral from 2 to 40 in the BHT-3000 and a numeral from 2 to 99 in the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, and they should satisfy the following condition:

mini.no.digits ≤ *max.no.digits*

If both of *mini.no.digits* and *max.no.digits* are omitted, the default reading range is 2 to 40 digits in the BHT-3000 and 2 to 99 digits in the BHT-4000/BHT-5000/BHT-6000. In the BHT-6500/BHT-7000/BHT-7500, if both of them are omitted, the default reading range is from the minimum number of digits specified in System Mode up to 99 digits.

If only *max.no.digits* is omitted, the BHT can only read the number of digits specified by *mini.no.digits*.

CD is a check digit. Specifying a `C` to *CD* makes the Interpreter check bar codes with MOD-10. The check digit is included in the number of digits.

```
OPEN "BAR:" AS #1 CODE "I:6-10C"
```

☐Codabar (NW-7) (N)

Syntax 1:

```
N[:[mini.no.digits[-max.no.digits]][startstop][CD]]
```

Syntax 2 (BHT-6500/BHT-7000/BHT-7500):

```
N[:[mini.no.digits[-max.no.digits]][startstop][CD]
 [,[mini.no.digits[-max.no.digits]][startstop][CD]]
 [,[mini.no.digits[-max.no.digits]][startstop][CD]]]
```

where

mini.no.digits and *max.no.digits* are the minimum and maximum numbers of digits for bar codes to be read by the BHT, respectively.

They should be a numeral from 3 to 40 in the BHT-3000 and a numeral from 3 to 99 in the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, and they should satisfy the following condition:

mini.no.digits ≤ *max.no.digits*

If both of *mini.no.digits* and *max.no.digits* are omitted, the default reading range is 3 to 40 digits in the BHT-3000 and 3 to 99 digits in the BHT-4000/BHT-5000/BHT-6000. In the BHT-6500/BHT-7000/BHT-7500, if both of them are omitted, the default reading range is from the minimum number of digits specified in System Mode up to 99 digits.

If only *max.no.digits* is omitted, the BHT can only read the number of digits specified by *mini.no.digits*.

start and *stop* are the start and stop characters, respectively. Each of them should be an A, B, C, or *D*. If a question mark (?) is specified, it acts as a wild card. The start and stop characters are included in the number of digits. The A through D will be stored in the barcode buffer as a through d.

CD is a check digit. Specifying a C to *CD* makes the Interpreter check bar codes with MOD-16. The check digit is included in the number of digits.

```
OPEN "BAR:" AS #1 CODE "N:8AAC"
```

□Code 39 (M)

Syntax 1:

```
M[:[mini.no.digits[-max.no.digits]][CD]]
```

Syntax 2 (BHT-6500/BHT-7000/BHT-7500):

```
M[:[mini.no.digits[-max.no.digits]][CD]
 [,[mini.no.digits[-max.no.digits]][CD]]
 [,[mini.no.digits[-max.no.digits]][CD]]]
```

where
mini.no.digits and *max.no.digits* are the minimum and maximum numbers of digits for bar codes to be read by the BHT, respectively.

They should be a numeral from 1 to 40 in the BHT-3000 and a numeral from 1 to 99 in the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, excluding start/stop characters. They should satisfy the following condition:

mini.no.digits ≤ *max.no.digits*

If both of *mini.no.digits* and *max.no.digits* are omitted, the default reading range is 1 to 40 digits in the BHT-3000 and 1 to 99 digits in the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500. If only *max.no.digits* is omitted, the BHT can only read the number of digits specified by *mini.no.digits*.

CD is a check digit. Specifying a C to *CD* makes the Interpreter check bar codes with MOD-43. The check digit is included in the number of digits.

```
OPEN "BAR:" AS #1 CODE "M:8-12C"
```

□Code 93 (L)

Syntax 1:

```
L[:[mini.no.digits[-max.no.digits]]]
```

Syntax 2 (BHT-6500/BHT-7000/BHT-7500):

```
L[:[mini.no.digits[-max.no.digits]
 [,[mini.no.digits[-max.no.digits]]
 [,[mini.no.digits[-max.no.digits]]]
```

where
mini.no.digits and *max.no.digits* are the minimum and maximum numbers of digits for bar codes to be read by the BHT, respectively.

They should be a numeral from 1 to 40 in the BHT-3000 and a numeral from 1 to 99 in the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, excluding start/stop characters and check digits. They should satisfy the following condition:

mini.no.digits ≤ *max.no.digits*

If both of *mini.no.digits* and *max.no.digits* are omitted, the default reading range is 1 to 40 digits in the BHT-3000 and 1 to 99 digits in the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500. If only *max.no.digits* is omitted, the BHT can only read the number of digits specified by *mini.no.digits*.

```
OPEN "BAR:" AS #1 CODE "L:6-12"
```

Neither start/stop characters nor check digits will be transferred to the barcode buffer.

☐**Code 128 (K)**

Syntax 1:

```
K[:[mini.no.digits[-max.no.digits]]]
```

Syntax 2 (BHT-6500/BHT-7000/BHT-7500):

```
K[:[mini.no.digits[-max.no.digits]]
 [,[mini.no.digits[-max.no.digits]]]
 [,[mini.no.digits[-max.no.digits]]]]
```

where

mini.no.digits and *max.no.digits* are the minimum and maximum numbers of digits for bar codes to be read by the BHT, respectively.

They should be a numeral from 1 to 40 in the BHT-3000 and a numeral from 1 to 99 in the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, excluding start/stop characters and check digit. They should satisfy the following condition:

mini.no.digits ≤ *max.no.digits*

If both of *mini.no.digits* and *max.no.digits* are omitted, the default reading range is 1 to 40 digits in the BHT-3000 and 1 to 99 digits in the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500. If only *max.no.digits* is omitted, the BHT can only read the number of digits specified by *mini.no.digits*.

```
OPEN "BAR:" AS #1 CODE "K:6-12"
```

Neither start/stop characters nor check digit will be transferred to the barcode buffer.

If the BHT reads any bar code consisting of special characters only (such as FNC, CODEA, CODEB, CODEC and SHIFT characters), it will not transfer the data to the barcode buffer.

FNC characters will be handled as follows:

(1) FNC1

The BHT-3000/BHT-4000/BHT-5000/BHT-6000 will not transfer FNC1 characters to the barcode buffer at all.

The BHT-6500/BHT-7000/BHT-7500 will not transfer an FNC1 character placed at the first or second character position immediately following the start character, to the barcode buffer. FNC1 characters in any other positions will be converted to GS characters (1Dh) and then transferred to the barcode buffer like normal data.

In the BHT-5000/BHT-6500/BHT-7000/BHT-7500, if an FNC1 immediately follows the start character, the bar code will be recognized as EAN-128 and marked with `W` instead of `K`.

(2) FNC2

If the BHT reads a bar code containing an FNC2 character(s), it will not buffer such data but transfer it excluding the FNC2 character(s).

(3) FNC3

If the BHT-3000/BHT-4000/BHT-5000/BHT-6000 reads a bar code containing an FNC3 character(s), it will transfer it excluding the FNC3 character(s), to the barcode buffer.

If the BHT-6500/BHT-7000/BHT-7500 reads a bar code containing an FNC3 character(s), it will regard the data as invalid and transfer no data to the barcode buffer, while it may drive the indicator LED and beeper (vibrator) if activated with the `OPEN` statement.

(4) FNC4

If the BHT-3000/BHT-4000/BHT-5000/BHT-6000 reads a bar code containing an FNC4 character(s), it will transfer it excluding the FNC4 character(s), to the barcode buffer.

In the BHT-6500/BHT-7000/BHT-7500, an FNC4 converts data encoded by the code set A or B into a set of extended ASCII-encoded data (128 added to each official ASCII code value).

A single FN4 character converts only the subsequent data character into the extended ASCII-encoded data.

A pair of FNC4 characters placed in successive positions converts all of the subsequent data characters preceding the next pair of FNC4 characters or the stop character, into the extended ASCII-encoded data. If a single FNC4 character is inserted in those data characters, however, it does not convert the subsequent data character only.

An FNC4 character does not convert any of GS characters converted by an FNC1 character, into the extended ASCII-encoded data.

☐**Standard 2 of 5 (STF) (H)** (For the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500)

Syntax 1:

```
H[:[mini.no.digits[-max.no.digits]][CD][start-
 stop]]
```

Syntax 2 (BHT-6500/BHT-7000/BHT-7500):

```
H[:[mini.no.digits[-max.no.digits]][CD] [start-
 stop]
 [,[mini.no.digits[-max.no.digits]][CD] [start-
 stop]]
 [,[mini.no.digits[-max.no.digits]][CD] [start-
 stop]]]
```

where

mini.no.digits and *max.no.digits* are the minimum and maximum numbers of digits for bar codes to be read.

They should be a numeral from 1 to 99 (excluding start/stop characters) and satisfy the following condition:

$mini.no.digits \leq max.no.digits$

If both of *mini.no.digits* and *max.no.digits* are omitted, the default reading range is from 1 to 99 digits in the BHT-4000/BHT-5000/BHT-6000 and from the minimum number of digits specified in System Mode up to 99 digits in the BHT-6500/BHT-7000/BHT-7500.

If only *max.no.digits* is omitted, only the number of digits specified by *mini.no.digits* can be read.

CD is a check digits. Specifying a C to *CD* makes the Interpreter check bar codes with MOD-10. The check digit is included in the number of digits.

startstop specifies the normal or short format of the start/stop characters. Specify N for the normal format; specify S for the short format. If *startstop* is omitted, start/stop characters can be read in either format.

```
OPEN "BAR:" AS #1 CODE "H:6-12"
```

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	The number of the specified read codes exceeds eight.

Run-time errors:

Error code	Meaning
`02h`	Syntax error (*`readcode`* is missing.)
`05h`	Parameter out of the range (*`readcode`* is not correct.)
`37h`	File already open
`3Ah`	File number out of the range
`45h`	Device files prohibited from opening concurrently (You attempted to open the bar code device file and the optical interface of the communications device file concurrently in the BHT-6000/BHT-6500/BHT-7000/BHT-7500.)

OPEN "COM:"

Opens a communications device file.

Syntax:

Syntax 1 (For the BHT-3000 and the direct-connect interface of the BHT-6000/BHT-6500/BHT-7000/BHT-7500):

```
OPEN "COMn:[baud][,[parity][,[charalength][,[stop-
bit][,[RS/CS][,[timeout]]]]]] "AS [#] filenumber
```

Syntax 2 (For the BHT-4000):

```
OPEN "COMn:[baud][,[parity][,[charalength][,[stopbit]
[,[RS/CS][,[timeout][,[RS][,[ER]]]]]]]] "AS [#] filenum-
ber
```

Syntax 3 (For the high-speed transmission in the BHT-4000):

```
OPEN "COMn:HS" AS [#] filenumber
```

Syntax 4 (For the BHT-5000):

```
OPEN "COMn:[baud][,[parity][,[charalength][,[stopbit]
[,[RS/CS][,[timeout][,[RS]]]]]]] "AS [#] filenumber
```

Syntax 5 (For the optical interface of the BHT-6000/BHT-6500/BHT-7000/BHT-7500):

```
OPEN "COMn: [baud] "AS [#] filenumber
```

Parameter:

baud

BHT-3000/BHT-4000/BHT-5000	38400*, 19200, 9600, 4800, 2400, 1200, 600, or 300 (*In the BHT-3000/BHT-4000, 38400 is supported by the direct-connect interface only)
BHT-6000/BHT-6500	(For the optical interface) 115200, 57600, 38600, 19200, 9600, or 2400 (For the direct-connect interface) 38400, 19200, 9600, 4800, 2400, 1200, 600, or 300
BHT-7000/BHT-7500	(For the optical interface) 115200, 57600, 38400, 19200, 9600, or 2400 (For the direct-connect interface) 115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200, 600, or 300

parity

N, E, or O

charalength

8 or 7

stopbit

1 or 2

RS/CS

0, 1, 2, 3 or 4

timeout

An integer numeral from 0 to 255.

RS

0 or 1

ER

0 or 1

filenumber

A numeric expression which returns a value from 1 to 16.

Description:

`OPEN "COM:"` opens a communications device file and associates it with *filenumber* for allowing input/output activities using the communications interface.

- If optional parameters enclosed with brackets are omitted, the most recently specified values or the defaults become active.

  Listed below are the defaults:

Baud rate	9600 bps
Parity check	No parity
Character length	8 bits
Stop bit	1 bit
RS/CS control	0 (No control)
Timeout	3 seconds
RS control*[1]	1 (Enabled)
ER control*[2]	1 (Enabled)

*[1] Supported by the BHT-4000 or by the optical interface of the BHT-5000.

*[2] Supported by the direct-connect interface of the BHT-4000.

■ COM*n*

`COM`*n* is a communications device file name.

For the BHT-3000 which supports both the optical and direct-connect interfaces and can open them concurrently, you can set both "COM1:" and "COM2:".

For the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500 which supports both the optical and direct-connect interfaces but cannot open them concurrently, you should set one of the specifications listed above. If you attempt to open both interfaces concurrently, a run-time error will occur.

Interface	Communications device file name
Optical interface	"COM1:"
Direct-connect interface	"COM2:"
Default interface[*3]	"COM:"

[*3] The default interface refers to an interface which is selected on the Set Com menu (BHT-3000), on the SET COM ENVIRONMENT menu (BHT-4000), on the SET COMMUNICATION menu (BHT-5000/BHT-7000/BHT-7500), or on the SET COM menu (BHT-6000/BHT-6500) in System Mode. (For details, refer to the BHT User's Manual.)

The BHT-6000/BHT-6500/BHT-7000/BHT-7500 cannot open the optical interface and the bar code device file concurrently. If you attempt to open them concurrently, a run-time error will occur.

`COM` may be in lowercase as shown below.

```
OPEN "com:" AS #8
```

■ *baud*

In the BHT-3000/BHT-4000/BHT-5000, *baud* is one of the baud rates: 38400*, 19600, 9600 (default), 4800, 2400, 1200, 600, and 300. (*The 38400 bps is supported by the direct-connect interface of the BHT-3000/BHT-4000 and by the BHT-5000.)

In the BHT-6000/BHT-6500, when the optical interface is used, *baud* is one of the baud rates: 115200, 57600, 38400. 19200, 9600 (default), and 2400. When the direct-connect interface is used, it is one of the baud rates: 38400, 19200, 9600 (default), 4800, 2400, 1200, 600, and 300.

In the BHT-7000/BHT-7500, when the optical interface is used, *baud* is one of the baud rates: 115200, 57600, 38400, 19200, 9600 (default), and 2400. When the direct-connect interface is used, it is one of the baud rates: 115200, 57600, ,38400, 19200 (default), 9600, 4800, 2400, 1200, 600, and 300.

■ *parity*

parity is a parity check. It should be `N` (default), `E`, or `O`, which corresponds to None, Even, or Odd parity, respectively.

■ *charalength*

charalength is a character length or the number of data bits. It should be `8` (default) or `7` bits.

■ *stopbit*

`stopbit` is the number of stop bits. It should be `1` (default) or `2` bits.

NOTE The optical interface of the BHT-6000/BHT-6500/BHT-7000/BHT-7500 is compliant with the IrDA physical layer (IrDA-SIR1.0), so the vertical parity, character length, and stop bit length are fixed to none, 8 bits, and 1 bit, respectively. If selected, those parameters will be ignored.

■ *RS/CS*

RS/CS enables or disables the RS/CS control. It should be `0` (default), `1`, `2`, `3`, or `4`, which corresponds to the following function:

Value of *RS/CS*	BHT-3000/BHT-6000/ BHT-6500/BHT-7000/ BHT-7500		BHT-4000		BHT-5000	
	Optical I/F	Direct-connect I/F	Optical I/F	Direct-connect I/F	Optical I/F	Direct-connect I/F
0 (default)	Ignored		RS/CS control disabled		RS/CS control disabled	Ignored
1	Ignored		RS/CS control enabled		RS/CS control enabled	Ignored
2	Ignored	High RD will be regarded as a high CS.	Run-time error		Run-time error	High RD will be regarded as high CS.
3	Ignored	Low RD will be regarded as high CS.	Run-time error		Run-time error	Low RD will be regarded as high CS.
4	Ignored	CS control disabled (RD will be used as an input port.)	Run-time error		Run-time error	CS control disabled (RD will be used as an input port.

As listed above, you can specify *RS/CS* option for the direct-connect interface of the BHT-3000/BHT-6000/BHT-6500/BHT-7000/BHT-7500 and for the BHT-4000/BHT-5000. If you specify it for the optical interface of the BHT-3000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, it will be ignored resulting in no run-time error.

RS/CS option is also applicable to Busy control when the direct-connect interface is used in the BHT-3000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500. To do so, interface cable connection should be modified. For details, refer to the BHT User's Manual.

Shown below is a coding sample for enabling the RS/CS control.

```
OPEN "COM:,,,,1" AS #16
```

Instead of the `OPEN "COM:"` statement, you can use an `OUT` statement for controlling the RS signal (supported by the optical interface of the BHT-5000 and by the BHT-4000) or the ER signal (which is supported by the BHT-4000). Also, you can use a `WAIT` statement or `INP` function for monitoring the CS signal or CD signal (supported by the BHT-4000). (To connect the BHT to an asynchronous half-duplex modem, it is necessary to use the `OUT` and `WAIT` statements and `INP` function.)

■ *timeout*

timeout is a maximum waiting time length until the CS signal goes ON after the BHT becomes ready to send data. It should be 0 to 255 in increment of 100 ms.

Specification of zero (0) causes no timeout.

Timeout is supported by the optical interface of the BHT-5000 and by the BHT-4000. Shown below is a coding sample for setting 10 seconds to *timeout*.

```
OPEN "COM:,,,,1,100" AS #6
```

To make the direct-connect interface of the BHT-3000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500 support timeout, the *RS/CS* option should be set to "2" or "3" so that the RD signal is regarded as CS. If any of "0," "1," and "4" has been set to the *RS/CS* option, the value of the *timeout* option will be modified.

The optical interface of the BHT-3000/BHT-6000/BHT-6500/BHT-7000/BHT-7500 does not support timeout. If specified, the *timeout* option will be ignored resulting in no run-time error.

■ *RS* (For the BHT-4000/BHT-5000)

RS specifies whether the RS signal should go ON or OFF when the `OPEN "COM:"` statement opens a communications device file of the optical interface in the BHT-4000/BHT-5000. You should set 0 (OFF) or 1 (ON: default). This specification is effective only when the RS/CS control is disabled.

■ *ER* (For the BHT-4000)

ER specifies whether the ER signal should go ON or OFF when the `OPEN "COM:"` statement opens a communications device file in the BHT-4000. You should set 0 (OFF) or 1 (ON: default). This specification is effective only when the direct-connect interface is selected. If specified for the optical interface, the *ER* option will be ignored resulting in no run-time error. In the BHT-3000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, specifying this *ER* option results in a run-time error.

■ *HS* (High-speed transmission for the BHT-4000)

This specification is effective only in the BHT-4000. In other BHTs, specifying HS results in a run-time error.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	*`filenumber`* is missing.

Run-time errors:

Error code	Meaning
`02h`	Syntax error (The `x` in `"COM:x"` contains an invalid parameter.)
`37h`	File already open
`3Ah`	File number out of the range
`45h`	File already open (You attempted to open the bar code device file and the optical interface of the communications device file concurrently in the BHT-6000/BHT-6500/BHT-7000/BHT-7500.) (You attempted to open the wireless interface and optical interface of the communications device file, or the wireless interface and direct-connect interface concurrently in the BHT-7500.)

OUT

Sends a data byte to an output port.

Syntax:

`OUT` *`portnumber,data`*

Parameter:

`portnumber`

A numeric expression.

`data`

A numeric expression which returns a value from 0 to 255.

Description:

`OUT` sends a data byte designated by *`data`* to a port specified by *`portnumber`*.

- *`portnumber`* is not an actual hardware port number on the BHT but a logical one which the Interpreter assigns. (Refer to Appendix D, "I/O Ports.")
- If bits not assigned a hardware resource are specified to *`portnumber`* or *`data`*, they will be ignored.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• *`portnumber`* is missing. • *`data`* is missing.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`portnumber`* or *`data`* is out of the range.)

Example:

```
OUT  3,7
```

The above example sets the LCD contrast to the maximum.

Reference:

Statements:	`WAIT`
Functions:	`NP`

POWER

Controls the automatic power-off facility.

Syntax:

Syntax 1 (Turning off the power according to the power-off counter):

```
POWER counter
```

Syntax 2 (Turning off the power immediately):

```
POWER {OFF|0}
```

Syntax 3 (Disabling the automatic power-off facility):

```
POWER CONT
```

Parameter:

counter

A numeric expression which returns a value from 0 to 32767.

Description:

■ Turning off the power according to the power-off counter

`POWER` *counter* turns off the power after the length of time specified by *counter* from execution of the `POWER` statement.

- *counter* is a setting value of the power-off counter in seconds. Shown below is a sample program for turning off the power after 4800 seconds from execution of `POWER` statement.

```
POWER 4800
```

- If no `POWER` statement is issued, the default counter value is 180 seconds.
- If any of the following operations and events happens while the power-off counter is counting, the counter will be reset to the preset value and start counting again:
  - Any key is pressed.
  - The trigger switch is pressed.
  - The BHT sends or receives data via a communications device file. (If a communications device file is closed, this operation does not reset the power-off counter.)

■ Turning off the power immediately

Execution of `POWER OFF` or `POWER 0` immediately turns off the power.

- The execution of `POWER OFF` or `POWER 0` deactivates the resume function if preset.

■ Disabling the automatic power-off facility

`POWER CONT` disables the automatic power-off facility.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`counter`* is out of the range.)

PRINT

Displays data on the LCD screen.

Syntax:

`PRINT [`*`data`*`[`*`CR/LFcontrol`*`...]]`

Parameter:

`data`

A numeric or string expression.

`CR/LFcontrol`

A comma (,) or a semicolon (;).

Description:

`PRINT` displays a number or a character string specified by *`data`* at the current cursor position on the LCD screen (To position the cursor, use a `LOCATE` statement.) and then repositions the cursor according to *`CR/LFcontrol.`*

■ *`data`*

- *`data`* may be displayed in any of the screen modes (the single-byte ANK mode, two-byte Kanji mode, and condensed two-byte Kanji mode). (The condensed two-byte Kanji mode is supported by the BHT-4000/BHT-5000.) It is, however, necessary to select the screen mode by a `SCREEN` statement before execution of the `PRINT` statement.

  If you specify single-byte ANK characters for *`data`* after selecting the two-byte Kanji mode or condensed two-byte Kanji mode with a `SCREEN` statement, then those ANK characters will appear in the half-width size.

```
CLS
SCREEN 1          'Kanji mode
PRINT "ABC123"
SCREEN 0          'ANK mode
PRINT "DEF456"
```

hese statements produce this output:

```
ABC123
DEF456
```

- In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, *`data`* may be displayed in standard size or small size depending upon the display font size selected.
- If you omit *`data`* option, a blank line is outputted. That is, the cursor moves to the first column of the next screen line.
- Positive numbers and zero automatically display with a leading space.
- Control codes (08h to 1Fh) appear as a space, except for BS (08h), CR (0Dh) and C (18h) codes.

  BS (08h) deletes a character immediately preceding the cursor so that the cursor moves backwards by one column.

  ```
  PRINT CHR$(8);
  ```

  CR (0Dh) causes a carriage return so that the cursor moves to the first column of the next screen line.

  ```
  PRINT CHR$(&h0D);
  ```

  C (18h) clears the LCD screen so that the cursor moves to its home position in the top left corner, just like the `CLS` statement.

  ```
  PRINT CHR$(&h18);
  ```

■ *`CR/LFcontrol`*

`CR/LFcontrol` determines where the cursor is to be positioned after the `PRINT` statement executes.

- If *`CR/LFcontrol`* is a comma (,), the cursor moves to the column position of a least multiple of 8 plus one following the last character output.

  Statement example: `PRINT 123,`

  Output:

  ```
  ␣123␣␣␣␣_
  ```

- If *`CR/LFcontrol`* is a semicolon (;), the cursor moves to the column position immediately following the last character output.

  Statement example: `PRINT 123;`

  Output:

  ```
  123_
  ```

- If neither a comma (,) nor semicolon (;) is specified to *`CR/LFcontrol`*, the cursor moves to the first column on the next screen line.

  Statement example: `PRINT 123`

  Output:

  ```
  123
  _
  ```

  In any of the above cases, the screen automatically scrolls up so that the cursor always positions in view on the LCD screen.

  To extend one program line to more than 512 characters in a single `PRINT` statement, you should use an underline (_) preceding a CR code, not a comma (,) preceding a CR code.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	*`data`* contains a comma (,) or semicolon (;).

Reference:

Statements: `LOCATE`, `PRINT USING`, and `SCREEN`

File I/O statement

PRINT

Outputs data to a communications device file.

Syntax:

```
PRINT #filenumber[,data[CR/LFcontrol...]]
```

Parameter:

filenumber

A numeric expression which returns a value from 1 to 16.

data

A numeric or string expression.

CR/LFcontrol

A comma (,) or a semicolon (;).

Description:

`PRINT` # outputs a numeric value or a character string specified by *data* to a communications device file specified by *filenumber*.

■ *filenumber*

- *filenumber* is a communications device file number assigned when the file is opened.

■ *CR/LFcontrol*

- If *CR/LFcontrol* is a comma (,), the `PRINT #` statement pads data with spaces so that the number of data bytes becomes a least multiple of 8, before outputting the data.

Statement example: `PRINT #1,"ABC","123"`

Output: `ABC_ _ _ _ _123 CR LF` ("_" denotes a space.)

- If *`CR/LFcontrol`* is a semicolon (;), the `PRINT #` statement outputs data without adding spaces or control codes.

Statement example:	`PRINT #1,"ABC";"123";`
Output:	`ABC123`

- If neither a comma (,) nor semicolon (;) is specified to *`CR/LFcontrol`*, the `PRINT #` statement adds a CR and LF codes.

Statement example:	`PRINT #1,"ABC123"`
Output:	`ABC123 CR LF`

To extend one program line to more than 512 characters in a single `PRINT #` statement, you should use an underline (_) preceding a CR code, not a comma (,) preceding a CR code.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• *`filenumber`* is missing. • *`data`* contains a comma (,) or semicolon (;).

Run-time errors:

Error code	Meaning
`34h`	Bad file name or number (You specified *`filenumber`* of an unopened file.)
`36h`	Improper file type (You specified *`filenumber`* of a file other than communications device files.)
`3Ah`	File number out of the range

Reference:

Statements:	`OPEN`

I/O statement

PRINT USING

Displays data on the LCD screen under formatting control.

Syntax:

Syntax 1 (Displaying numbers):

```
PRINT USING "numericformat";expression[CR/LFcontrol
[expression]...]
```

Syntax 2 (Displaying strings):

```
PRINT USING "stringformat";stringexpression
[CR/LFcontrol[stringexpression]...]
```

Parameter:

numericformat

#, a decimal point (.), and/or +.

stringformat

!, @, and/or &

CR/LFcontrol

A comma (,) or a semicolon (;).

Description:

`PRINT USING` displays a number or a character string specified by *expression* or *stringexpression* on the LCD according to a format specified by *numericformat* or *stringformat*, respectively.

- To extend one program line to more than 512 characters in a single `PRINT USING` statement, you should use an underline (_) preceding a CR code, not a comma (,) preceding a CR code.

■ *numericformat*

numericformat is a formatting string consisting of #, decimal point (.), and/or +, each of which causes a special printing effect as described below.

# Represents a digit position.

If the number specified by *expression* has fewer digits than the number of digit positions specified by #, it is padded with spaces and right-justified.

Statement example: `PRINT USING "#####";123`

Output:

```
  123
```

If the number specified by *expression* has more digits than the number of digit positions specified by #, the extra digits before the decimal point are truncated and those after the decimal point are rounded.

Statement example: `PRINT USING "###.#";1234.56`

Output:

```
234.6
```

. Specifies the position of the decimal point.

If the number specified by *expression* has fewer digits than the number of digit positions specified by # after the decimal point, the insufficient digits appear as zeros.

Statement example: `PRINT USING "####.###";123`

Output:

```
 123.000
```

+ Displays the sign of the number.

If + is at the beginning of the format string, the sign appears before the number specified by *expression*; if + is at the end of the format string, the sign appears after the number. If the number specified by *expression* is a positive number or zero, it is preceded or followed by a space instead of a sign. (+)

Statement example: `PRINT USING "+#####";-123`

Output:

```
-123
```

■ *`stringformat`*

`stringformat` is a formatting string consisting of `!`, `@`, and/or &&, each of which causes a special printing effect as described below.

! Displays the first character of the *`stringexpression`*.

Statement example: `PRINT USING "!";"ABC"`

Output:

```
A
```

`@` Displays the entire *`stringexpression`*.

Statement example: `PRINT USING "@";"ABC"`

Output:

```
ABC
```

&& Displays the first n+2 characters of the *`stringexpression`*, where n is the number of spaces between the ampersands (&&).

If the format field specified by *`stringformat`* is longer than the *`stringexpression`*, the string is left-justified and padded with space; if it is shorter, the extra characters are truncated.

Statement example: `PRINT USING "&   &";"ABCDE"`

Output:

```
ABCDE
```

Below are statement examples containing incorrect formatting strings.

Example: `PRINT USING "Answer=###";a`

Example: `PRINT USING "####.# ######";a,b`

■ *`expression`* or *`stringexpression`*

If more than one number or string is specified, the `PRINT USING` statement displays each of them according to *`numericformat`* or *`stringformat`*, respectively.

```
PRINT USING "###";a,b,c
```

■ *`CR/LFcontrol`*

`CR/LFcontrol` determines where the cursor is to be positioned after the `PRINT USING` statement executes. For details, refer to the *`CR/LFcontrol`* in the `PRINT` statement.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• *`numericformat`* is not correct. • *`expression`* or *`stringexpression`* contains a comma (,) or semicolon (;).
`error 86: ';' missing`	No semicolon (;) follows `"`*`numericformat`*`"` or `"string-format"`.

Declarative statement

PRIVATE

Declares one or more work variables or register variables defined in a file, to be private.

Syntax:

Syntax 1:

`PRIVATE` *`varname`* `[,`*`varname`*`...]`

Syntax 2:

`PRIVATE DEFREG` *`registerdefinition`* `[,`*`registerdefinition`*`...]`

Parameter:

`varname`

`numericvariable` `[(`*`subscript`*`)]`

`stringvariable` `[(`*`subscript`*`)[[`*`stringlength`*`]]]`

`registerdefinition`

`non-arraynumericvariable` `[=`*`numericconstant`*`]`

`arraynumericvariable``(`*`subscript`*`)` `[=`*`numericinitialvaluedefinition`*`]`

`non-arraystringvariable``[[`*`stringlength`*`]]` `[=`*`stringconstant`*`]`

`arraystringvariable``(`*`subscript`*`)[[`*`stringlength`*`]]` `[=`*`stringinitialvaluedefinition`*`]`

`numericinitialvaluedefinition`

For one-dimensional:
`{`*`numericconstant`*`[,`*`numericconstant`*`...]}`

For two-dimensional:
`{{`*`numericconstant`*`[,`*`numericconstant`*`...]},` `{`*`numericconstant`*`[,`*`numericconstant`*`...]} ...}`

`stringinitialvaluedefinition`

For one-dimensional:
`{`*`stringconstant`*`[,`*`stringconstant`*`...]}`

For two-dimensional:
`{{`*`stringconstant`*`[,`*`stringconstant`*`...]},` `{`*`stringconstant`*`[,`*`stringconstant`*`...]} ...}`

subscript

For one-dimensional: *integerconstant*

For two-dimensional:
integerconstant,integerconstant

Where *integerconstant* is a numeric expression which returns a value from 0 to 254.

stringlength

An integer constant from 1 to 255.

Description:

PRIVATE allows variables declared by *varname* or *registerdefinition* to be referred to or updated in that file.

- Inside one PRIVATE statement, up to 30 variables can be declared to *varname* or *registerdefinition*.
- You may declare non-array variables and array variables together to *varname*.
- For details about *registerdefinition*, refer to DEFREG statement.

Syntax errors:

Error code and message		Meaning
error 7:	Variable name redefinition	The array declared with PRIVATE had been already declared with DEFREG.
error 71:	Syntax error	• *stringlength* is out of the range. • *stringlength* is not an integer constant.
error 72:	Variable name redefinition	• A same variable name is double declared inside a same PRIVATE statement. • A same variable name is used for a non-array variable and array variable.
error 78:	Array symbols exceed 30 for one DIM, PRIVATE, or GLOBAL statement	More than 30 variables are declared inside one PRIVATE statement.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range
`07h`	Insufficient memory space (The variable area has run out.)
`0Ah`	Duplicate definition (An array is double declared.)

Reference:

Statements: `DEFREG`, `DIM`, and `GLOBAL`

PUT

Writes a record from a field variable to a data file.

Syntax:

```
PUT [#]filenumber[,recordnumber]
```

Parameter:

filenumber

A numeric expression which returns a value from 1 to 16.

recordnumber

A numeric expression which returns a value from 1 to 32767.

Description:

PUT writes a record from a field variable(s) declared by the FIELD statement to a data file specified by *filenumber*.

- *filenumber* is the number of a data file opened by the OPEN statement.
- *recordnumber* is the record number where the data is to be placed in a data file.

  It should be within the range from 1 to the maximum number of registrable records (*filelength*) specified by the OPEN statement (when a new data file is created).
- If *recordnumber* option is omitted, the default record number is one more than the last record written.
- Record numbers to be specified do not have to be continuous. If you specify record number 10 when records 1 through 7 have been written, for example, the PUT statement automatically creates records 8 and 9 filled with spaces and then writes data to record 10.
- If the actual data length of a field variable is longer than the field width specified by the FIELD statement, the excess is truncated from the right end column.
- Since data in a data file is treated as text data (ASCII strings), numeric data should be converted into the proper string form with the STR$ function before being assigned to a field variable.
- In the BHT-5000/BHT-6000/BHT-6500, the PUT statement cannot write data to files stored in drive B.
- In the BHT-7000/BHT-7500, the PUT statement cannot write data to files opened as read-only by specifying drive B in the OPEN statement.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	*`filenumber`* is missing.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (• *`filenumber`* is out of the range.) (• *`recordnumber`* is out of the range.)
`07h`	Insufficient memory space
`34h`	Bad file name or number (You specified *`filenumber`* of an unopened file.)
`36h`	Improper file type (You specified *`filenumber`* of a file other than data files.)
`3Ah`	*`Filenumber`* out of the range
`3Eh`	A `PUT` statement executed without a `FIELD` statement.
`41h`	File damaged
`42h`	File write error (You attempted to write onto a read-only file.)
`43h`	Not allowed to access the data in drive B.

Reference:

Statements:	`GET` and `OPEN`

READ

Reads data defined by DATA statement(s) and assigns them to variables.

Syntax:

`READ` *`variable`*`[,`*`variable`*`...]`

Parameter:

`variable`

A numeric or string variable.

Description:

READ reads as many data values as necessary in turn from data stored by DATA statement and assigns them, one by one, to each variable in the READ statement.

- If the data type of a read value does not match that of the corresponding variable, the following operations take place so that no error occurs:

  - Assigning a numeric data to a string variable:

  The READ statement converts the numeric data into the string data type and then assigns it to the string variable.

  Statement example:

  ```
  DATA 123
  READ a$
  PRINT a$
  ```

  Output:

  ```
  123
  ```

  - Assigning a string data to a numeric variable:

  If the string data is valid as numeric data, the READ statement converts the string data into the numeric data type and then assigns it to the numeric variable.

  Statement example:

  ```
  DATA "123"
  READ b
  PRINT b
  ```

  Output:

  ```
  123
  ```

If the string data is invalid as numeric data, the READ statement assigns the value 0 to the numeric variable.

Statement example:

```
DATA "ABC"
READ c
PRINT c
```

Output:

```
0
```

- The number of data values stored by the DATA statement must be equal to or greater than that of variables specified by the READ statement. If not, a run-time error occurs.
- To specify the desired DATA statement location where the READ statement should start reading data, you use the RESTORE statement.

Run-time errors:

Error code	Meaning
04h	Out of DATA (No DATA values remain to be read by the READ statement.)

Reference:

Statements: DATA and RESTORE

REM

Declares the rest of a program line to be remarks or comments.

Syntax:

Syntax 1:

```
REM comment
```

Syntax 2:

```
' comment
```

Description:

`REM` causes the rest of a program line to be treated as a programmer's remark or comment for the sake of the program readability and future program maintenance. The remark statements are non-executable.

- Difference in description between syntax 1 and syntax 2:

  The keyword `REM` cannot begin in the first column of a program line. When following any other statement, `REM` should be separated from it with a colon (:).

  An apostrophe ('), which may be replaced for keyword `REM`, can begin in the first column. When following any other statement, an apostrophe (') requires no colon (:) as a delimiter.

- You can branch to a `REM` statement labelled by the `GOTO` or `GOSUB` statement. The control is transferred to the first executable statement following the `REM` statement.

Syntax errors:

Error code and message		Meaning
`error 2:`	`Improper label name` (redefinition, variable name, or reserved word used)	`REM` begins in the first column of a program line.

Reference:

Statements: `$INCLUDE`

I/O statement

RESTORE

Specifies a DATA statement location where the READ statement should start reading data.

Syntax:

```
RESTORE [label]
```

Description:

RESTORE specifies a DATA statement location where the READ statement should start reading data, according to *label* designating the DATA statement.

- You can specify DATA statements in included files.
- If *label* option is omitted, the default label is a DATA statement appearing first in the user program.

Syntax errors:

Error code and message	Meaning
`error 81: Must be DATA statement label`	*label* is not a DATA statement label.

Reference:

Statements:	DATA and READ

RESUME

Causes program execution to resume at a specified location after control is transferred to an error-handling routine.

Syntax:

Syntax 1:

```
RESUME [0]
```

Syntax 2:

```
RESUME NEXT
```

Syntax 3:

RESUME *label*

Description:

RESUME returns control from the error-handling routine to a specified location of the main program to resume program execution.

- The RESUME statement has three forms as listed below. The form determines where execution resumes.

RESUME or RESUME 0	Resumes program execution with the statement that caused the error.
RESUME NEXT	Resumes program execution with the statement immediately following the one that caused the error.
RESUME *label*	Resumes program execution with the statement designated by *label*.

- The RESUME statement should be put inside the error-handling routine.

Syntax errors:

Error code and message	Meaning
error 71: Syntax error	*label* has not been defined.

Run-time errors:

Error code	Meaning
`14h`	`RESUME` without error (`RESUME` statement occurs outside of an error-handling routine.)

Reference:

Statements:	`ON ERROR GOTO`
Functions:	`ERL` and `ERR`

RETURN

Returns control from a subroutine or an event-handling routine (for keystroke interrupt).

Syntax:

```
RETURN
```

Description:

`RETURN` statement in a subroutine returns control to the statement immediately following the `GOSUB` that called the subroutine.

`RETURN` statement in an event-handling routine for keystroke interrupt returns control to the program location immediately following the one where the keystroke trap occurred.

- No label designating a return location should be specified in a `RETURN` statement.
- You may specify more than one `RETURN` statement in a subroutine or an event-handling routine.

Reference:

Statements: `GOSUB` and `ON KEY...GOSUB`

I/O statement

SCREEN

Sets the screen mode and the character attribute.

Syntax:

Syntax 1:

`SCREEN` *`screenmode`*`[,`*`charaattribute`*`]`

Syntax 2:

`SCREEN ,`*`charaattribute`*

Parameter:

`screenmode` and *`charaattribute`*

A numeric expression which returns a value from 0 to 3.

Description:

`SCREEN` sets the screen mode and the character attribute of the LCD screen according to *`screenmode`* and *`charaattribute`*, respectively, as listed below.

Screen mode	*screenmode*	SCREEN statement
Single-byte ANK mode (default)	0	`SCREEN 0`
Two-byte Kanji mode	1	`SCREEN 1`
Condensed two-byte Kanji mode*	2	`SCREEN 2`

* The condensed two-byte Kanji mode is supported by the BHT-4000/BHT-5000. Specifying this mode in the BHT-3000/BHT-6000/BHT-6500/BHT-7000/BHT-7500 will result in a run-time error (05h).

Character attribute	*charaattribute*	SCREEN statement
Normal display (default)	0	`SCREEN ,0`
Highlighted display	1	`SCREEN ,1`
Normal display, double-width characters*	2	`SCREEN ,2`
Highlighted display, double-width characters*	3	`SCREEN ,3`

* Double-width is supported by the BHT-7000/BHT-7500. Specifying it in the BHT-3000/BHT-4000/BHT-5000/BHT-6000/BHT-6500 will result in a run-time error (05h).

- At program startup, the defaults--single-byte ANK mode and normal display--are active.
- If a parameter is omitted, the corresponding screen mode or character attribute does not change.
- In the two-byte Kanji mode, characters can be displayed in either the full-width size (16 dots wide by 16 dots high) or the half-width size (8 dots wide by 16 dots high). In the BHT-6000/BHT-6500/BHT-7000/BHT-7500, if the small-size font is selected, characters will be displayed in either the full-width size (12 dots wide by 12 dots high) or the half-width size (6 dots wide by 12 dots high).
- In the condensed two-byte Kanji mode (supported by the BHT-4000/BHT-5000), characters can be displayed in either the full-width size (12 dots wide by 16 dots high) or the half-width size (6 dots wide by 16 dots high).

Run-time errors:

Error code	Meaning
`02h`	Syntax error (BHT-3000: The two-byte Kanji mode is set by the `SCREEN` statement although you have selected the English message version on the Set Resume menu in System Mode.)
`05h`	Parameter out of the range

Flow control statement

SELECT...CASE...END SELECT

Conditionally executes one of statement blocks depending upon the value of an expression.

Syntax:

```
SELECT conditionalexpression
       CASE test1
              [statementblock]
       [CASE test2
              [statementblock]]...
       [CASE ELSE
              [statementblock]]
END SELECT
```

Parameter:

conditionalexpression, *test*1, and *test*2

A numeric or string expression.

Description:

This statement executes one of *statementblock*s depending upon the value of *conditionalexpression* according to the steps below.

(1) SELECT evaluates *conditionalexpression* and compares it with *test*s sequentially to look for a match.

(2) When a match is found, the associated *statementblock* executes and then control passes to the first statement following the END SELECT.

If no match is found, the *statementblock* following the CASE ELSE executes and then control passes to the first statement following the END SELECT. If you include no CASE ELSE, control passes to the first statement following the END SELECT.

- If the SELECT statement block includes more than one CASE statement containing the same value of *test*, only the first CASE statement executes and then control passes to the first statement following the END SELECT.
- If a CASE followed by no executable statement is encountered, control passes to the first statement following the END SELECT.
- *conditionalexpression* (numeric or string) and *test*s must agree in type.

- You can nest the SELECT...CASE...END SELECT statements to a maximum of 10 levels.

```
SELECT a
   CASE 1
      SELECT b
         CASE 3
            PRINT "a=1,b=3"
      END SELECT
   CASE 2
      PRINT "a=2"
END SELECT
```

- When using the SELECT...CASE statement block together with block-structured statements (DEF FN...END DEF, FOR...NEXT, FUNCTION...END FUNCTION, IF...THEN...ELSE...END IF, SELECT...CASE...END SELECT, SUB...END SUB and WHILE...WEND), you can nest them to a maximum of 30 levels.

Syntax errors:

Error code and message		Meaning
error 26:		Too deep nesting.
error 55:	Incorrect use of SELECT... CASE...END SELECT	CASE, CASE ELSE, or END SELECT statement appears outside of the SELECT statement block.
error 56:	Incomplete control structure	No END SELECT corresponds to SELECT.
error 71:	Syntax error	*conditionalexpression* and *tests* do not agree in type.

Run-time errors:

Error code	Meaning
0Ch	CASE and END SELECT without SELECT
10h	Expression too long or complex (The program nesting by SELECT statement block is too deep.)

User-defined function statement

SUB...END SUB

Names and defines user-created function SUB.

Syntax:

Syntax 1 (Defining a numeric function):

`SUB` *`subname`* `[(`*`dummyparameter`*`[,`*`dummyparameter`*`]...)]`

Syntax 2 (Exiting from the function block prematurely):

`EXIT SUB`

Syntax 3 (Ending the function block):

`END SUB`

Syntax 4 (Calling a function):

`[CALL]` *`subname`*`[(`*`realparameter`*`[,`*`realparameter`*`]...)]`

Parameter:

`subname`

Real function name

`dummyparameter`

A non-array integer variable, a non-array real variable, or a non-array string variable.

`realparameter`

A numeric or string expression.

Description:

■ Creating a user-defined function

`SUB`...`END SUB` creates a user-defined function. The function definition block between `SUB` and `END SUB` is a set of some statements and functions.

- You cannot make double definition to a same function name.
- This statement block should not be defined in the block-structured statements (`DEF FN`...`END DEF`, `FOR`...`NEXT`, `FUNCTION`...`END FUNCTION`, `IF`...`THEN`...`ELSE`...`END IF`, `SELECT`...`CASE`...`END SELECT`, `SUB`...`END SUB`, and `WHILE`...`WEND`), in the error-handling routine, event-handling routine, or in the subroutines.
- `SUB`...`END SUB` functions can be recursive.
- *dummyparameter*, which corresponds to the variable having the same name in the function definition block, is a local variable valid only in that block. Therefore, if a variable having the same name as *dummyparameter* is used outside `SUB`...`END SUB` statement block or used as a *dummyparameter* of any other function in the same program, it will be independently treated.
- In user-defined functions, you can call other user-defined functions. You can nest `SUB`...`END SUB` statements to a maximum of 10 levels.
- When using the `SUB`...`END SUB` together with block-structured statements (`DEF FN`...`END DEF`, `FOR`...`NEXT`, `FUNCTION`...`END FUNCTION`, `IF`...`THEN`...`ELSE`...`END IF`, `SELECT`...`CASE`...`END SELECT`, `SUB`...`END SUB`, and `WHILE`...`WEND`), you can nest them to a maximum of 30 levels.
- If variables other than *dummyparameter*(s) are specified in the function definition block, they will be treated as local variables whose current values are available only in that function definition block, unless `PRIVATE` or `GLOBAL` is specified.
- `EXIT SUB` exits the function block prematurely and returns control to the position immediately after the statement that called the user-defined function.
- Unlike other user-defined functions, `SUB` function cannot assign a return value.

■ Calling a user-defined function

`CALL` statement and *subname* call a user-defined function. `CALL` can be omitted.

- The number of *realparameter*s should be equal to that of *dummyparameter*s, and the types of the corresponding variables used in those parameters should be identical.
- If you specify a global variable in *realparameter* when calling a user-defined function, the user-defined function cannot update the value of the global variable. This is because all *realparameter*s are passed not by address but by value. (So called "Call-by-value")

NOTE Before any call to a SUB...END SUB function, you need to place definition of the SUB function or declaration of the SUB by the DECLARE statement in your source program.

A function name is defined globally. If more than one same function name exists in a same project, therefore, a multiple symbol definition error will occur when files will be linked. The same error will occur also if the SUB...END SUB defines a user-created function in a file to be included and more than one file in a same project reads that included file.

Syntax errors:

■ When defining a user function

Error code and message		Meaning
`error 64:`	`Function redefinition`	You made double definition to a same function name.
`error 71:`	`Syntax error`	• The string length is out of the range. • The string length is not an integer constant.
`error 92:`	`Incorrect use of SUB, EXIT SUB or END SUB`	• The EXIT SUB statement is specified outside the function definition block. • The END SUB statement is specified outside the function definition block.
`error 93:`	`Incomplete control structure (SUB...END SUB)`	END SUB is missing.
`error 94:`	`Cannot use SUB in control structure`	The SUB...END SUB statement is defined in other block-structured statements such as FOR and IF statement blocks.

■ When calling a user-defined function

Error code and message	Meaning
`error 68: Mismatch argument type or number`	• The number of the real parameters is not equal to that of the dummy parameters. • *`dummyparameter`* was an integer variable in defining a function, but *`realparameter`* is a real type in calling the function. (If *`dummyparameter`* was a real variable in defining a function and *`realparameter`* is an integer type, then no error occurs.)
`error 69: Function undefined`	Calling of a user-defined function precedes the definition of the user-created function.

Run-time errors:

Error code	Meaning
`07h`	Insufficient memory space (You nested SUB statements to more than 10 levels.)
`0Fh`	String length out of the range (The returned value of the string length exceeds the allowable range.)

Reference:

Statements: `DECLARE`

Example:

File 1

```
DECLARE SUB add(x,y)
A=1:B=2
PRINT "TEST"
CALL add(A,B)
    ⋮
```

File 2

```
SUB add(X,Y)
PRINT X+Y
END SUB
```

```
TEST
3
```

I/O statement

WAIT

Pauses program execution until a designated input port presents a given bit pattern.

Syntax:

`WAIT` *portnumber*,*ANDbyte*[,*XORbyte*]

Parameter:

portnumber

A numeric expression

ANDbyte and *XORbyte*

A numeric expression which returns a value from 0 to 255.

Description:

`WAIT` suspends a user program while monitoring the input port designated by *portnumber* until the port presents the bit pattern given by *ANDbyte* and *XORbyte*. (Refer to Appendix D, "I/O Ports.")

Each bit in *ANDbyte* corresponds to a port bit you want to turn on. Each bit in *XORbyte* corresponds to a port bit you want to turn off.

The byte at the input port is first XORed with the *XORbyte* parameter. Next, the result is ANDed with the value of *ANDbyte* parameter.

If the final result is zero (0), the `WAIT` statement rereads the input port and continues the same process. If it is nonzero, control passes to the statement following the `WAIT`.

- If *XORbyte* option is omitted, the `WAIT` statement uses a value of zero (0).

```
WAIT 1,x  ' = WAIT 1,x,0
```

- If an invalid port number or bit data is specified, it will be assumed as zero (0) so that the `WAIT` statement may fall into an infinite loop.

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• *`portnumber`* is missing. • *`ANDbyte`* is missing.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range

Example:

```
WAIT 0,&H03
```

The above statement suspends a user program until any data is inputted from the keyboard or the bar code reader.

Reference:

Statements:	`OUT`
Functions:	`INP`

Flow control statement

WHILE...WEND

Continues to execute a statement block as long as the conditional expression is true.

Syntax:

```
WHILE conditionalexpression
           [statementblock]
WEND
```

Description:

A WHILE...WEND continues to execute *statementblock* as long as the *conditionalexpression* is true (not zero) according to the steps below.

(1) The *conditionalexpression* in the WHILE statement is evaluated.

(2) If the condition is false (zero), the *statementblock* is bypassed and control passes to the first statement following the WEND.

If the condition is true (not zero), the *statementblock* is executed. When WEND statement is encountered, control returns to the WHILE statement. (Go back to step (1) to be repeated.)

- The WHILE and WEND statements cannot be written on a same program line.
- If no WEND statement is written corresponding to the WHILE, a syntax error occurs.
- The BHT-BASIC does not support a DO...LOOP statement block.
- You can nest the WHILE...END statements to a maximum of 10 levels.
- When using the WHILE...WEND statement together with block-structured statements (DEF FN...END DEF, FOR...NEXT, FUNCTION...END FUNCTION, IF...THEN...ELSE...END IF, SELECT...CASE...END SELECT, SUB...END SUB, and WHILE...WEND), you can nest them to a maximum of 30 levels.

```
WHILE a
     WHILE b
          WHILE c
            ⋮
          WEND
     WEND
WEND
```

Syntax errors:

Error code and message		Meaning
`error 26:`		Too deep nesting.
`error 57:`	`Incorrect use of WHILE...WEND`	`WEND` appears outside of the `WHILE` statement block.
`error 58:`	`Incomplete control structure`	No `WEND` corresponds to `WHILE`.

Reference:

Statements: `FOR...NEXT`

I/O statement

XFILE

Transmits a designated file according to the specified communications protocol.

Syntax:

`XFILE "[`*`drivename`*`:]`*`filename`*`"[,"`*`protocolspec`*`"]`

Parameter:

`"[`*`drivename`*`:]`*`filename`*`"` and `"`*`protocolspec`*`"`

String expressions.

Description:

`XFILE` transmits a data file designated by `"[`*`drivename`*`:]`*`filename`*`"` between the BHT and host computer or between BHTs according to the communications protocol specified by `"`*`protocolspec`*`."` (For the BHT-protocol, refer to the BHT User's Manual. For the BHT-Ir protocol, refer to the "BHT-6000 User's Manual," "BHT-6500 User's Manual," "BHT-7000 User's Manual," or "BHT-7500 User's Manual.")

■ `"[`*`drivename`*`:]`*`filename`*`"`

`filename` is a data file name. For the format of data file names, refer to `OPEN` statement.

For the BHT-5000/BHT-6000/BHT-6500, the *`drivename`* may be `A:` or `B:`. If the *`drivename`* is omitted, the default `A:` applies.

In the BHT-7000/BHT-7500, the *`drivename`* (`A:` or `B:`) will be ignored.

■ `"`*`protocolspec`*`"`

`"`*`protocolspec`*`"` parameter can specify the following protocol specifications:

Specifications	BHT-protocol	BHT-Ir protocol	Multilink protocol
Transmission direction	√	√	√
Serial number	√		
Horizontal parity checking (BCC)	√		
Transmission monitoring	√	√	√
Handling of space codes in the tail of a data field during file transmission[*1]	√	√	√
Timeout length when a link will be established[*1]	√	√	
Checking whether filenames are identical[*2]	√	√	

[*1] Supported by the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500.

[*2] Supported by the BHT-7000/BHT-7500.

- Transmission direction

Parameter omitted (default)	Transmits a file from the BHT.
`R` or `r`	Receives a file from the host computer or any other BHT.

Example: `XFILE "d2.dat","R"`

`"[`*`drivename`*`:]`*`filename`*`"` cannot be omitted even in file reception.

- Serial number

Parameter omitted (default)	No serial number setting.
`S` or `s`	Adds a serial number to every transmission block.

Example: `XFILE "d2.dat","S"`

A serial number immediately follows a text control character heading each transmission block. It is a 5-digit decimal number. When it is less than five digits, the upper digits having no value are filled with zeros.

- Horizontal parity checking (BCC)

Parameter omitted (default)	No horizontal parity checking.
`P` or `p`	Suffixes a BCC to every transmission block.

Example: `XFILE "d2.dat","P"`

A block check character (BCC) immediately follows a terminator of each transmission block. The horizontal parity checking checks all bits except for headers (SOH and STX).

- Transmission monitoring

Parameter omitted (default)	No serial number indication.
`M` or `m`	Displays a serial number of the transmission block during file transmission.

Example: `XFILE "d2.dat","M"`

A serial number will appear in the 5-digit decimal format at the current cursor position before execution of the `XFILE` statement.

- Handling of space codes in the tail of a data field during file transmission (for the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500)

Parameter omitted (default)	Ignores space codes.
`T` or `t`	Handles space codes as data.

Example: `XFILE "d2.dat","T"`

Space codes placed in the tail of a data field will be handled as 20h in file reception.

- Timeout length when a link will be established (for the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500)

  Specify the timeout length by 1 to 9.

Set value	Downloading	Uploading	
		BHT-protocol	BHT-Ir protocol
1	30 sec.	Retries of ENQ, 10 times	Retries of ENQ, 60 times
2	60 sec.	Retries of ENQ, 20 times	Retries of ENQ, 120 times
3	90 sec.	Retries of ENQ, 30 times	Retries of ENQ, 180 times
4	120 sec.	Retries of ENQ, 40 times	Retries of ENQ, 240 times
5	150 sec.	Retries of ENQ, 50 times	Retries of ENQ, 300 times
6	180 sec.	Retries of ENQ, 60 times	Retries of ENQ, 360 times
7	210 sec.	Retries of ENQ, 70 times	Retries of ENQ, 420 times
8	240 sec.	Retries of ENQ, 80 times	Retries of ENQ, 480 times
9	No timeout	No timeout	No timeout

  Example: `XFILE "d2.dat","2"`

  In file reception, the timeout length is 60 seconds; in file transmission, the maxmum number of ENQ retries is 20 (when the BHT-protocol is used.)

- Checking whether filenames are identical (BHT-7000/BHT-7500)

  This option can apply only to file reception (that is, when the transmission direction is specified with `R` or `r`).

Parameter omitted (default)	Receives only a data file having the same name as specified in *`filename`*. The "*`filename`*" should be the same as that used in the sending station.
`N` or `n`	No checking whether filenames are identical. The BHT may receive a data file with a different name (given in the sending station) from that specified by *`filename`*. That is, the received file is renamed as specified by *`filename`*. If *`filename`* is omitted (only "" is specified), the BHT receives a data file with the name as is in the sending station.

Example: If a file is named "TEST.DAT" in the sending station

```
Sample 1.  XFILE "TEST.DAT","RN"  'Receives TEST.DAT as
                                  'TEST2.DAT.
Sample 2.  XFILE "","RN"          'Receives the file
                                  'with the same name
                                  'as used in the sending
                                  'station.
```

- A communications device file should be opened before execution of the `XFILE` statement. (For the file opening, refer to the `OPEN "COM:"` statement.)
- The `XFILE` statement uses the interface specified by the `OPEN "COM:"` statement.

  (If an `XFILE` statement is executed in the BHT-3000, not the interface specified by the `OPEN "COM:"` statement but the interface selected for the BHT-BASIC on the Set Com menu in System Mode will become active.)
- A data file to be transmitted should be closed beforehand.
- To transfer a file by using the BHT-Ir protocol or multilink protocol, set the BHT's ID to any of 1 to FFFFh. Specifying zero (0) to the ID will result in a run-time error.
- Undefined letters, if specified in *`protocolspec`*, will be ignored. The specifications below, therefore, produce the same operation. The last one of the timeout values goes active.

```
"RSPMT1"
"R,S,P,M,T,1"
"r,s,p,m,t,1"
"ABCDEFGHIJKLMNOPQRSTUVWXYZ1"

"2"
"3462"
"22"
```

- If you transmit a data file having the same name as that already used in the receiving station:
  - the newly transmitted file replaces the old one when the field structure is matched.
  - a run-time error occurs when the field structure is not matched.

  To receive a data file having the same name at the BHT but having a different structure, therefore, it is necessary to delete that old file.
- Pressing the Clear key during file transmission aborts the execution of the `XFILE` statement by issuing an `EOT` code and displays a run-time error.

Syntax errors:

Error code and message	Meaning
`error 3: '"' missing`	No double quote precedes or follows `[`*`drivename`*`:]`*`filename`*.
`error 71: Syntax error`	`[`*`drivename`*`:]`*`filename`* is not enclosed in double quotes.

Run-time errors:

Error code	Meaning
02h	Syntax error ([*drivename*:]*filename* is not correct.)
07h	Insufficient memory space (During file reception, the memory runs out.)
32h	File type mismatch (The received file is not a data file.)
33h	Received text format not correct
34h	Bad file name or number (You specified *filename* of an unopened file.)
35h	File not found
37h	File already open
38h	The file name is different from that in the receive header.
3Bh	The number of the records is greater than the defined maximum value.
3Eh	FIELD statement not executed yet
40h	ID not set
44h	No empty area of the specified size in the RAM
46h	Communications error (A communications protocol error has occurred.)
47h	Abnormal end of communications or termination of communications by the Clear key (The Clear key has aborted the file transmission.)
49h	Received program file not correct

Example:

The sample below transmits a data file by adding a serial number and horizontal parity checking, and then displays the serial number at the 1st line of the screen.

```
CLOSE
OPEN "d0.dat"AS #1
FIELD #1,10 AS A$,20 AS B$
L%=LOF(1)
CLOSE
LOCATE 1,1
PRINT "00000/";RIGHT$("00000"+MID$(STR$(L%),2),5)
LOCATE 1,1
OPEN "COM:19200,N,8,1" AS #8
XFILE "d0.dat","SPM"
CLOSE #8
```

Before file transmission → After file transmission

```
00000/00100        00100/00100
```

Reference:

Statements: OPEN and OPEN "COM:"

$INCLUDE

Specifies an included file.

Syntax:

Syntax 1:

```
REM $INCLUDE:'filename'
```

Syntax 2:

```
'$INCLUDE:'filename'
```

Description:

`$INCLUDE` reads a source program specified by *`'filename'`* into the program line immediately following the `$INCLUDE` line in compilation.

Storing definitions of variables, subroutines, user-defined functions, and other data to be shared by source programs into the included files will promote application of valuable program resources.

- *`filename`* is a file to be included.
- If the specified filename does not exist in compiling a source program, a fatal error occurs and the compilation terminates.
- No characters including space should be put between `$` and `INCLUDE` and between single quotes (') and *`filename`*.
- As shown below, if any character except for space or tab codes is placed between `REM` and `$INCLUDE` in syntax 1 or between a single quote (') and `$INCLUDE` in syntax 2, the program line will be regarded as a comment line so that the `$INCLUDE` statement will not execute.

  ```
  REM xxx $INCLUDE:'mdlprg1.SRC'
  ```

- Before specifying included files, it is necessary to debug them carefully.
- `$INCLUDE` statements cannot be nested.
- The program lines in included files are not outputted to the compile list.

  If a compilation error occurs in an included file, the error message shows the line number where the `$INCLUDE` statement is described.

  Symbols defined in included files are not outputted to the symbol list.
- If a program line in an included file refers to a variable, user-defined function, or others defined outside the included file, then the program line number where the `$INCLUDE` statement is described is outputted to the cross reference list, as the referred-to line.

Fatal Error:

Error code and message		Meaning
`fatal error 30:`	`Cannot find include file "XXX"`	No included file is found.
`fatal error 31:`	`Cannot nest include file`	Included files are nested.

Additional Explanation for Statements

■ Effective range of labels

Labels are effective only in a file.

■ Definition of common variables (by `COMMON` statement)

In an object to be executed first (that is, in a main object), you should define all common variables to be accessed. In any other objects, declare common variables required only in each object. If a first executed object is linked with an object where an undefined common variable(s) is newly defined, an error will result.

■ Definition and initialization of register variables (by `DEFREG` statement)

As for work variables, you should declare required register variables in each object. You may specify an initial value to a register variable in each object; however, giving different initial values to a same register variable in more than one object will result in an error in linking process.

Chapter 15
Function Reference

CONTENTS

ABSolute Numeric function

ABS

Returns the absolute value of a numeric expression.

Syntax:

```
ABS(numericexpression)
```

Description:

ABS returns the absolute value of `numericexpression`. The absolute value is the magnitude of `numericexpression` without regard to sign. For example, both ABS `(-12.34)` and ABS `(12.34)` are equal to 12.34.

- If you give a real number, this function returns a real number; if an integer number, this function returns an integer number.

ASCii code String function

ASC

Returns the ASCII code value of a given character.

Syntax:

`ASC(`*`stringexpression`*`)`

Description:

ASC returns the ASCII code value of the first character of *`stringexpression`*, which is an integer from 0 to 255. (For the ASCII character codes, refer to Appendix C, "Character Sets.")

- If *`stringexpression`* is a null string, this function returns the value 0.
- If given a two-byte Kanji character, this function cannot return the two-byte Kanji code.

Reference:

Functions: `CHR$`

Block Check Character String function

BCC$

Returns a block check character (BCC) of a data block.

Syntax:

`BCC$(`*`datablock`*`,`*`checktype`*`)`

Parameter:

`datablock`

A string expression.

`checktype`

A numeric expression which returns a value from 0 to 2.

Description:

BCC$ calculates a block check character (BCC) of *`datablock`* according to the block checking method specified by *`checktype`*, and returns the BCC.

- *`checktype`* is 0, 1, or 2 which specifies SUM, XOR, or CRC-16, respectively, as described below.

checktype	Block checking method	No. of charas for BCC	BCC	Generative polynomial
0	SUM	1	Lowest one byte of the sum of all character codes contained in a *`datablock`*.	
1	XOR	1	One byte gained by XORing all character codes contained in a *`datablock`*.	
2	CRC-16	2*	Two bytes gained rom the cyclic redundancy check operation applied to bit series of all characters in *`datablock`* with the bit order in each byte inverted.	$X^{16}+X^{15}+X^{2}+1$

*The upper byte and the lower byte of the operation result will be set to the 1st and 2nd characters, respectively.

- A common use for BCC$ is to perform block checking or to generate a BCC for a data block.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`checktype`* is out of the range.)

CHecK DiGiT String function

CHKDGT$

Returns a check digit of bar code data.

Syntax:

`CHKDGT$(`*`barcodedata`*`,`*`CDtype`*`)`

Parameter:

`barcodedata` and *`CDtype`*

String expressions.

Description:

`CHKDGT$` calculates a check digit (CD) of *`barcodedata`* according to the calculation method specified by *`CDtype`*, and then returns it as one-character string.

- *`CDtype`* is `A`, `H`, `I`, `M` or `N`, which specifies the bar code type and the corresponding calculation method as listed below.

CDtype	Bar Code Type	Calculation Method
A	EAN and UPC	MOD-10 (Modulo arithmetic-10)
H*	STF (Standard 2 of 5)	MOD-10 (Modulo arithmetic-10)
I	ITF (Interleaved 2 of 5)	MOD-10 (Modulo arithmetic-10)
M	Code 39	MOD-43 (Modulo arithmetic-43)
N	Codabar (NW-7)	MOD-16 (Modulo arithmetic-16)

* Supported by the BHT-7000/BHT-7500 only.

`CDtype` may be in lowercase.

- In the BHT-7000/BHT-7500, if *`barcodedata`* contains a character(s) out of the specification of the bar code type specified by *`CDtype`*, `CHKDGT$` returns a null string. However, if only the CD position character in *`barcodedata`* is out of the specification, `CHKDGT$` calculates the correct CD and returns it as one-character string.

Sample coding 1:	`CD.Data$=CHKDGT$("494AB4458","A")`
	"A" and "B" are out of the specification of the EAN or UPC, so `CD.Data$` will become a null string.
Sample coding 2:	`CD.Data$=CHKDGT$("4940045X","A")`
	"X" is a CD position character, so `CHKDGT$` calculates the correct CD and `CD.Data$` will become "8."
Sample coding 3:	`CD.Data$=CHKDGT$("a0ef3-a","N")`
	"e" and "f" are out of the specification of the Codabar (NW-7), so `CD.Data$` will become a null string.

Sample coding 4: `CD.Data$=CHKDGT$("a123Qa","N")`

"Q" is a CD position character, so `CHKDGT$` calculates the correct CD and `CD.Data$` will become "-."

■ When *`CDtype`* is A (EAN or UPC), `CHKDGT$` identifies the EAN or UPC of *`barcodedata`* depending upon the data length (number of digits) as listed below.

Data length of *`barcodedata`*	Universal Product Codes
13	EAN-13 or UPC-A
8	EAN-8
7	UPC-E

If the data length is a value other than 13, 8, and 7, this function returns a null string.

- To check that the CD is correct:

Pass a CD-suffixed *`barcodedata`* to a `CHKDGT$` as shown below. If the returned value is equal to the CD, the CD data is suitable for the *`barcodedata`*.

Sample coding:
```
IF CHKDGT$("49400458","A")="8"
THEN...
```

- To add a CD to barcode data:

Pass *`barcodedata`* followed by a dummy character to a `CHKDGT$` as shown below. The returned value will become the CD to be replaced with the dummy character.

Sample coding:
```
PRINT"4940045"+CHKDGT$("4940045"+"0","A")
```

```
49400458
```

■ When *`CDtype`* is `H` (STF), the length of *`barcodedata`* must be two or more digits. If not, `CHKDGT$` returns a null string.

- To check that the CD is correct:

Pass a CD-suffixed *`barcodedata`* to a `CHKDGT$` as shown below. If the returned value is equal to the CD, the CD data is suitable for the *`barcodedata`*.

Sample coding:
```
IF CHKDGT$("12345678905","H")="5"
THEN...
```

- To add a CD to barcode data:

Pass *`barcodedata`* followed by a dummy character to a `CHKDGT$` as shown below. The returned value will become the CD to be replaced with the dummy character.

Sample coding:
```
PRINT
"1234567890"+CHKDGT$("1234567890"+"0"."H")
```

```
12345678905
```

- When *`CDtype`* is `I` (ITF), the length of *`barcodedata`* must be an even number of two or more digits. If not, `CHKDGT$` returns a null string.

- To check that the CD is correct:

Pass a CD-suffixed *`barcodedata`* to a `CHKDGT$` as shown below. If the returned value is equal to the CD, the CD data is suitable for the *`barcodedata`*.

Sample coding:

```
IF CHKDGT$("123457","I")="7"
THEN...
```

- To add a CD to barcode data:

Pass *`barcodedata`* followed by a dummy character to a `CHKDGT$` as shown below. The returned value will become the CD to be replaced with the dummy character.

Sample coding:

```
PRINT "12345"+CHKDGT$("12345"+"0","I")
```

```
123457
```

- When *`CDtype`* is `M` (Code 39), the length of *`barcodedata`* must be two or more digits except for start and stop characters. If not, `CHKDGT$` returns a null string.

- To check that the CD is correct:

Pass a CD-suffixed *`barcodedata`* to a `CHKDGT$` as shown below. If the returned value is equal to the CD, the CD data is suitable for the *`barcodedata`*.

Sample coding:

```
IF CHKDGT$("CODE39W","M")="W"
THEN...
```

- To add a CD to barcode data:

Pass *`barcodedata`* followed by a dummy character to a `CHKDGT$` as shown below. The returned value will become the CD to be replaced with the dummy character.

Sample coding:

```
PRINT "CODE39"+CHKDGT$("CODE39"+"0","M")
```

```
CODE39W
```

■ When `CDtype` is `N` (Codabar), the length of *barcodedata* must be three digits or more including start and stop characters. If not, `CHKDGT$` returns a null string.

- To check that the CD is correct:

Pass a CD-suffixed *barcodedata* to a `CHKDGT$` as shown below. If the returned value is equal to the CD, the CD data is suitable for the *barcodedata*.

Sample coding:
```
IF CHKDGT$("a0123-a","N")="-"
THEN...
```

- To add a CD to barcode data:

Pass *barcodedata* followed by a dummy character and enclosed with start and stop characters, to a `CHKDGT$` as shown below. The returned value will become the CD to be replaced with the dummy character.

Sample coding:
```
ld%=LEN("a0123a")
PRINT LEFT$("a0123a",ld%-1)+CHKDGT$
("a01230a","N")+RIGHT$("a0123a",1)
```

```
a0123-a
```

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*CDtype* is out of the range.)

Reference:

Statements: `OPEN "BAR:"`

CHaRacter code String function

CHR$

Returns the character corresponding to a given ASCII code.

Syntax:

`CHR$(`*`characode`*`)`

Parameter:

`characode`

A numeric expression which returns a value from 0 to 255.

Description:

`CHR$` converts a numerical ASCII code specified by *`characode`* into the equivalent single-byte character. This function is used to send control codes (e.g., ENQ and ACK) to a communications device file or to display a double quotation mark or other characters having special meanings in the BHT-BASIC.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`characode`* is out of the range.)

Example:

- To output an ACK code to a communications device file, use `CHR$(&H06)`. The ASCII value for the ACK code is &H06.

```
PRINT #1,CHR$(&H06);
```

- To display control codes from 8 (08h) to 31 (1Fh), refer to the program examples shown in the `PRINT` statement.
- To display double quotation marks around a string, use `CHR$(34)` as shown below. The ASCII value for a double quotation mark is 34 (22h).

```
PRINT CHR$(34);"Barcode";CHR$(&H22)
```

```
"Barcode"
```

- To display a Kanji code, use a shift JIS code as shown below. The shift JIS code for 漢 is 8ABFh.

```
SCREEN 1
PRINT CHR$(&h8A);CHR$(&hBF)
```

```
漢
```

Reference:

Statements:	`PRINT`
Functions:	`ASC`

COUNTRY I/O function

COUNTRY$

Sets a national character set or returns a current country code.

Syntax:

Syntax 1 (Setting a national character set):

`COUNTRY$="`*`countrycode`*`"`

Syntax 2 (Returning a country code):

`COUNTRY$`

Parameter:

`countrycode`

A string expression--A, D, E, F, G, I, J, N, S, or W

Description:

■ Syntax 1

`COUNTRY$` sets a national character set specified by "*countrycode*". The national character set is assigned to codes from 32 (20h) to 127 (7Fh). (Refer to Appendix C2, "National Character Sets.")

- "*countrycode*" specifies one of the following national character sets:

countrycode	National character set
A	America (default)
D	Denmark
E	England
F	France
G	Germany
I	Italy
J	Japan (default)
N	Norway
S	Spain
W	Sweden

- After setting a national character set, you may display it for codes from 32 (20h) to 127 (7Fh), on the LCD.
- If `"`*`countrycode`*`"` is omitted, the default national character set is America (code A) or Japan (code J) when you have selected the English or Japanese message version on the menu screen* in System Mode, respectively.

  * Menu screen for selecting the message version

BHT Series	Menu screen
BHT-3000	Set Resume menu
BHT-4000/BHT-5000/BHT-6000/ BHT-6500/BHT-7000/BHT-7500	SET DISPLAY menu

- `"`*`countrycode`*`"` set by this function remains effective in the programs chained by `CHAIN` statements.
- If `"`*`countrycode`*`"` has more than one character, only the first one takes effect.
- If `"`*`countrycode`*`"` is an invalid letter other than those listed above, the function is ignored.
- `"`*`countrycode`*`"` may be in lowercase.

```
        COUNTRY$="j"
```

■ Syntax 2

`COUNTRY$` returns a current country code as an uppercase alphabetic letter.

CurSoR LINe I/O function

CSRLIN

Returns the current row number of the cursor.

Syntax:

```
CSRLIN
```

Description:

`CSRLIN` returns the current row number of the cursor as an integer, in the current screen mode selected by a `SCREEN` statement.

Screen mode		BHT-3000	BHT-4000	BHT-5000	BHT-6000/ BHT-6500	BHT-7000	BHT-7500
Single-byte ANK mode	Standard-size font	1 to 4	1 to 10 (1 to 9*)	1 to 8	1 to 6	1 to 8	1 to 20
	Small-size font	–	–	–	1 to 8	1 to 10	1 to 26
Two-byte Kanji mode	Standard-size font	1 to 3	1 to 9 (1 to 8*)	1 toto 7	1 to 5	1 to 7	1 to 19
	Small-size font	–	–	–	1 to 7	1 to 9	1 to 25
Condensed two-byte Kanji mode		–	1 to 9 (1 to 8*)	1 to 7	–	–	–

* When the system status is displayed on the LCD.

- Even if the cursor is invisible (by a `LOCATE` statement), the `CSRLIN` function operates.
- For the current column number of the cursor, refer to the `POS` function.

Reference:

Statements:	`LOCATE` and `SCREEN`
Functions:	`POS`

DATE$

Returns the current system date or sets a specified system date.

Syntax:

Syntax 1 (Retrieving the current system date):

`DATE$`

Syntax 2 (Setting the current system date):

`DATE$="`*`date`*`"`

Parameter:

`date`

A string expression.

Description:

■ Syntax 1

`DATE$` returns the current system date as an 8-byte string. The string has the format below.

`yy/mm/dd`

where *`yy`* is the lower two digits of the year from 00 to 99, *`mm`* is the month from 01 to 12, and *`dd`* is the day from 01 to 31.

■ Syntax 2

`DATE$` sets the system date specified by `"`*`date`*`"`. The format of `"`*`date`*`"` is the same as that in syntax 1.

Example: `date$="00/10/12"`

- The year `yy` must be the lower two digits of the year: otherwise, the system does not compensate for leap years automatically.
- The calendar clock is backed up by the battery. (For the system time, refer to the `TIME$` function.)

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`date`* is out of the range.)

Reference:

Functions:	`TIME$`

EOF

Tests whether the end of a device I/O file has been reached.

Syntax:

`EOF([#]`*`filenumber`*`)`

Parameter:

`filenumber`

A numeric expression which returns a value from 1 to 16.

Description:

`EOF` tests for an end of a device I/O file designated by *`filenumber`*. Then it returns -1 (true) if no data remains; it returns 0 (false) if any data remains, as listed below.

File Type	Returned Value	End-of-file Condition
Communications device file	-1 (true)	No data remains in the receive buffer.
	0 (false)	Any data remains in the receive buffer.
Barcode device file	-1 (true)	No data remains in the barcode buffer
	0 (false)	Any data remains in the barcode buffer.

- *`filenumber`* should be the file number of an opened device file.
- The `EOF` function cannot be used for data files. Specifying a data file number for *`filenumber`* causes a run-time error.

Run-time errors:

Error code	Meaning
`34h`	Bad file name or number (You specified *`filenumber`* of an unopened file.)
`36h`	Improper file type (You specified *`filenumber`* of a data file.)
`3Ah`	File number out of the range

Reference:

Statements:	`INPUT#`, `LINE INPUT#`, `OPEN "BAR:"`, and `OPEN "COM:"`
Functions:	`INPUT$`, `LOC`, and `LOF`

ERror Line Error-handling function

ERL

Returns the current statement location of the program where a run-time error occurred.

Syntax:

```
ERL
```

Description:

ERL returns the current statement location of the program where a run-time error occurred most recently.

- The ERL function works only with line numbers and not with labels.
- The returned value is in decimals, so it may be necessary to use the HEX$ function for decimal-to-hexadecimal conversion when using the ERL function in error-handling routines.
- Addresses which the ERL returns correspond to ones that are outputted to the left end of the address-source list in hexadecimals when a +L option is specified in compilation, if converted from decimals to hexadecimals with the HEX$ function.
- Since the ERL function returns a significant value only when a run-time error occurs, you should use this function in error-handling routines where you can check the error type for effective error recovery.

Reference:

Statements:	ON ERROR GOTO and RESUME
Functions:	ERR and HEX$

ERRor code Error-handling function

ERR

Returns the error code of the most recent run-time error.

Syntax:

```
ERR
```

Description:

ERR returns the code of a run-time error that invoked the error-handling routine.

- The returned value is in decimals, so it may be necessary to use the HEX$ function for decimal-to-hexadecimal conversion when using the ERR function in error-handling routines.
- Codes which the ERR returns correspond to ones that are listed in Appendix A1, "Run-time Errors," if converted from decimals to hexadecimals with the HEX$ function.
- Since the ERR function returns a significant value only when a run-time error occurs, you should use this function in error-handling routines where you can check the error type for effective error recovery.

Reference:

Statements:	ON ERROR GOTO and RESUME
Functions:	ERL and HEX$

ETX$

Modifies the value of a terminator (ETX) for the BHT-protocol; also returns the current value of a terminator.

Syntax:

Syntax 1 (Changing the value of a terminator):

`ETX$=`*`stringexpression`*

Syntax 2 (Returning the current value of a terminator):

`ETX$`

Parameter:

`stringexpression`

A string expression which returns a single-byte character.

Description:

■ Syntax 1

`ETX$` modifies the value of a terminator (one of the text control characters) which indicates the end of data text in the BHT-protocol when a data file is transmitted by an `XFILE` statement. (For the BHT-protocol, refer to the BHT User's Manual.)

- ETX$ is called a protocol function.
- The initial value of a terminator (ETX) is 03h.

■ Syntax 2

ETX$ returns the current value of a terminator.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`stringexpression`* is a null string.)
`0Fh`	String length out of the range (*`stringexpression`* is more than a single byte.)

Reference:

Statements:	`OPEN "COM:"` and `XFILE`
Functions:	`SOH$` and `STX$`

FREe area Memory management function

FRE

Returns the number of bytes available in a specified area of the memory.

Syntax:

```
FRE(areaspec)
```

Parameter:

areaspec

A numeric expression which returns a value from 0 to 3.

Description:

FRE returns the number of bytes left unused in a memory area specified by *areaspec* listed below.

areaspec	Memory area
0	Array work variable area
1	File area
2	Operation stack area for the Interpreter
3	File area in drive B (in the BHT-5000/BHT-6000/BHT-6500)

- The file area will be allocated to data files and program files in cluster units. The FRE function returns the total number of bytes of non-allocated clusters. (For details about a cluster, refer to Appendix F, "Memory Area.")
- The operation stack area for the Interpreter is mainly used for numeric operations, string operations, and for calling user-defined functions.
- A returned value of this function is a decimal number.

Run-time errors:

Error code	Meaning
05h	Parameter out of the range (*areaspec* is out of the range.)

HEX$

Converts a decimal number into the equivalent hexadecimal string.

Syntax:

```
HEX$(numericexpression)
```

Parameter:

numericexpression

A numeric expression which returns a value from -32768 to 32767.

Description:

HEX$ function converts a decimal number from -32768 to 32767 into the equivalent hexadecimal string which is expressed with 0 to 9 and A to F.

Listed below are conversion examples.

numericexpression	Returned value
-32768	8000
-1	FFFF
0	0
1	1
32767	7FFF

Run-time errors:

Error code	Meaning
06h	The operation result is out of the allowable range.

INput KEYboard | I/O function

INKEY$

Returns a character read from the keyboard.

Syntax:

`INKEY$`

Description:

`INKEY$` reads from the keyboard to see whether a key has been pressed, and returns one character read. If no key has been pressed, `INKEY$` returns a null string. (For the character codes, refer to Appendix C. For the key number assignment, refer to Appendix E.)

- `INKEY$` does not echo back a read character on the LCD screen.
- A common use for `INKEY$` is to monitor a keystroke while the BHT is ready for bar code reading or other events.
- If any key previously specified for keystroke trapping is pressed, `INKEY$` cannot return the typed data since the `INKEY$` has lower priority than keystroke trapping.
- To display the cursor, you use the `LOCATE` and `CURSOR` statements as shown below.

```
LOCATE,,1:CURSOR ON
k$=INKEY$
IF k$="" THEN...
```

Reference:

Statements:	`CURSOR, KEY OFF, KEY ON,` and `LOCATE`
Functions:	`ASC` and `INPUT$`

INP

Returns a byte read from a specified input port.

Syntax:

`INP(`*`portnumber`*`)`

Parameter:

`portnumber`

A numeric expression which returns a value from 0 to 32767.

Description:

`INP` reads one-byte data from an input port specified by *`portnumber`* and returns the value. (For the input port numbers, refer to Appendix D, "I/O Ports.")

- If you specify an invalid value to *`portnumber`*, `INP` returns an indeterminate value.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`portnumber`* is out of the range.)

Reference:

Statements: `OUT` and `WAIT`

INPUT File I/O function

INPUT$

Returns a specified number of characters read from the keyboard or from a device file.

Syntax:

Syntax 1 (Reading from the keyboard):

`INPUT$(`*`numcharas`*`)`

Syntax 2 (Reading from a device file):

`INPUT$(`*`numcharas`*`,[#]`*`filenumber`*`)`

Parameter:

`numcharas`

A numeric expression which returns a value from 1 to 255.

`filenumber`

A numeric expression which returns a value from 1 to 16.

Description:

`INPUT$` reads the number of characters specified by *`numcharas`* from the keyboard or from a device file specified by *`filenumber`*, then returns the resulting string.

■ Syntax 1 (without specification of *`filenumber`*)

`INPUT$` reads a string or control codes from the keyboard.

- `INPUT$` does not echo back read characters on the LCD screen.
- The cursor shape (invisible, underlined, or full block) depends upon the specification selected by the `LOCATE` statement.
- If any key previously specified for keystroke trapping is pressed during execution of the `INPUT$`, the keyboard input will be ignored; that is, neither typed data is read by `INPUT$` nor keystroke is trapped.

■ Syntax 2 (with specification of *`filenumber`*)

`INPUT$` reads from a device file (the bar code device file or any of the communications device files).

- The number of characters in a device file can be indicated by using a `LOC` function.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`numcharas`* is out of the range.)
`34h`	Bad file name or number (You specified *`filenumber`* of an unopened file.)
`36h`	Improper file type (You specified *`filenumber`* of a data file.)
`3Ah`	File number out the range

Reference:

Statements:	`CURSOR`, `INPUT`, `LINE INPUT`, `LOCATE`, `OPEN "BAR:"`, and `OPEN "COM:"`
Functions:	`EOF`, `INKEY$`, `LOC`, and `LOF`

IN STRing String function

INSTR

Searches a specified target string for a specified search string, and then returns the position where the search string is found.

Syntax:

`INSTR([`*`startposition`*`,]`*`targetstring`*`,`*`searchstring`*`)`

Parameter:

`startposition`

A numeric expression which returns a value from 1 to 32767.

`targetstring` and *`searchstring`*

A string expression.

Description:

`INSTR` searches a target string specified by *`targetstring`* to check whether a search string specified by *`searchstring`* is present in it, and then returns the first character position of the search string first found.

- *`startposition`* is the character position where the search is to begin in *`targetstring`*. If you omit *`startposition`* option, the search begins at the first character of *`targetstring`*.
- *`targetstring`* is the string being searched.
- *`searchstring`* is the string you are looking for.

NOTE Do not mistake the description order of *`targetstring`* and *`searchstring`*.

- A returned value of `INSTR` is a decimal number from 0 to 255, depending upon the conditions as listed below.

Conditions	Returned value
If *`searchstring`* is found within *`targetstring`*:	First character position of the search string first found
If *`startposition`* is greater than the length of *`targetstring`* or 255:	0
If *`targetstring`* is a null string:	0
If *`searchstring`* is not found:	0
If *`searchstring`* is a null string:	Value of *`startposition`* 1 if *`startposition`* option is omitted.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`startposition`* is out of the range.)

Reference:

Functions: `LEN`

INTeger — Numeric operation function

INT

Returns the largest whole number less than or equal to the value of a given numeric expression

Syntax:

`INT(`*`numericexpression`*`)`

Parameter:

`numericexpression`

A real expression.

Description:

`INT` returns the largest whole number less than or equal to the value of *`numericexpression`* by stripping off the fractional part.

- You use `INT` as shown below to round off the fractional part of a real number.

  `INT(`*`realnumber`*`+0.5)`

  Example:
  ```
  dat=1.5
  PRINT INT(dat+0.5)
  ```
  ```
  2
  ```

- If *`numericexpression`* is negative, this function operates as shown below.

  ```
  PRINT INT(-1.5)
  PRINT INT(-0.2)
  ```
  ```
  -2
  -1
  ```

LEFT$

Returns the specified number of leftmost characters from a given string expression.

Syntax:

`LEFT$(`*`stringexpression`*`,`*`stringlength`*`)`

Parameter:

`stringlength`

A numeric expression which returns a value from 0 to 255.

Description:

`LEFT$` extracts a portion of a string specified by *`stringexpression`* by the number of characters specified by *`stringlength`*, starting at the left side of the string.

- If *`stringlength`* is zero, `LEFT$` returns a null string.
- If *`stringlength`* is greater than the length of *`stringexpression`*, the whole *`stringexpression`* will be returned.

Run-time errors:

Error code	Meaning
05h	Parameter out of the range (*`stringlength`* is out of the range.)

Reference:

Functions:	`LEN, MID$,` and `RIGHT$`

LENgth String function

LEN

Returns the length (number of bytes) of a given string.

Syntax:

`LEN(`*`stringexpression`*`)`

Description:

`LEN` returns the length of *`stringexpression`*, that is, the number of bytes in the range from 0 to 255.

- If *`stringexpression`* is a null string, `LEN` returns the value 0.
- `LEN` counts a full-width Kanji (in the two-byte code mode) as two characters.

```
PRINT LEN("漢字")
```

```
4
```

LOC

Returns the current position within a specified file.

Syntax:

```
LOC([#]filenumber)
```

Parameter:

filenumber

A numeric expression which returns a value from 1 to 16.

Description:

`LOC` returns the current position within a file (a data file, communications device file, or bar code device file) specified by *filenumber*.

- Depending upon the file type, the content of the returned value differs as listed below.

File type	Returned value
Data file	Record number following the number of the last record read by a `GET` statement
Communications device file	Number of characters contained in the receive buffer (0 if no data is present in the receive buffer.)
Bar code device file	Number of characters contained in the bar-code buffer* (0 if the BHT is waiting for bar code reading.)

* The size of the barcode buffer is 40 bytes in the BHT-3000, and 99 bytes in the BHT-4000/BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500.

- If `LOC` is used before execution of the first `GET` statement after a data file is opened, it returns 1 or 0 when the data file has any or no data, respectively.

Run-time errors:

Error code	Meaning
`34h`	Bad file name or number (You specified *`filenumber`* of an unopened file.)
`3Ah`	File number out of the range
`3Eh`	A `PUT` or `GET` statement executed without a `FIELD` statement. (No `FIELD` statement is found.)

Reference:

Statements:	`OPEN`
Functions:	`EOF` and `LOF`

LOF

Returns the length of a specified file.

Syntax:

```
LOF([#]filenumber)
```

Parameter:

filenumber

A numeric expression which returns a value from 1 to 16.

Description:

`LOF` returns the length of a data file or communications device file specified by *filenumber*.

- Depending upon the file type, the content of the returned value differs as listed below.

File type	Returned value
Data file	Number of written records
Communications device file	Number of bytes of unoccupied area in the receive buffer

- If you specify the bar code device file, a run-time error will occur.

Run-time errors:

Error code	Meaning
34h	Bad file name or number (You specified *filenumber* of an unopened file.)
36h	Improper file type (You specified *filenumber* of a bar code device file.)
3Ah	File number out of the range

Reference:

Statements:	`GET`, `INPUT`, `LINE INPUT`, `OPEN`, and `OPEN "COM:"`
Functions:	`EOF`, `INPUT$`, and `LOC`

code MARK I/O function

MARK$

Returns a bar code type and the number of digits of the bar code.

Syntax:

```
MARK$
```

Description:

`MARK$` returns a 3-byte string which consists of the first one byte representing a bar code type and the remaining two bytes indicating the number of digits of the bar code.

- The first one byte of a returned value contains one of the following letters representing bar code types:

Bar code type	First one byte of a returned value
EAN-13 or UPC-A	A
EAN-8	B
UPC-E	C
ITF (Interleaved 2 of 5)	I
STF (Standard 2 of 5)	H
Codabar (NW-7)	N
Code 39	M
Code 93	L
Code 128	K
EAN-128	W

- The remaining two bytes of a returned value indicate the number of digits of the bar code in decimal notation.
- `MARK$` returns a null string until bar code reading takes place first after start of the program.

MID$

Returns a portion of a given string expression from anywhere in the string.

Syntax:

`MID$(`*`stringexpression`*`,`*`startposition`*`[,`*`stringlength`*`])`

Parameter:

`startposition`

A numeric expression which returns a value from 1 to 255.

`stringlength`

A numeric expression which returns a value from 0 to 255.

Description:

Starting from a position specified by *`startposition`*, `MID$` extracts a portion of a string specified by *`stringexpression`*, by the number of characters specified by *`stringlength`*.

- A returned value of `MID$` depends upon the conditions as listed below.

Conditions	Returned value
If *`stringlength`* option is omitted:	All characters from *`startposition`* to the end of the string Example: `PRINT MID$("ABC123",3)` `C123`
If *`stringlength`* is greater than the number of characters contained between *`startposition`* and the end of the string:	All characters from *`startposition`* to the end of the string Example: `PRINT MID$("ABC123",3,10)` `C123`
If *`startposition`* is greater than the length of *`stringexpression`*:	Null string Example: `PRINT MID$("ABC123",10,1)`

NOTE BHT-BASIC does not support such MID$ function that replaces a part of a string variable.

Run-time errors:

Error code	Meaning
05h	Parameter out of the range

Reference:

Functions: LEFT$, LEN, and RIGHT$

POS

Returns the current column number of the cursor.

Syntax:

```
POS(0)
```

Description:

POS returns the current column number of the cursor in the current screen mode selected by a SCREEN statement, as an integer.

Screen mode		BHT-3000	BHT-4000	BHT-5000	BHT-6000/ BHT-6500	BHT-7000	BHT-7500
Single-byte ANK mode	Standard-size font	1 to 17	1 to 27	1 to 22	1 to 17	1 to 22	1 to 27
	Small-size font	–	–	–	1 to 17	1 to 22	1 to 27
Two-byte Kanji mode	Standard-size font	1 to 13	1 to 21	1 to 17	1 to 13	1 to 17	1 to 21
	Small-size font	–	–	–	1 to 17	1 to 22	1 to 27
Condensed two-byte Kanji mode		–	1 to 27	1 to 22	–	–	–

- Even if the cursor is invisible (by a LOCATE statement), the POS function operates.
- If the maximum value in the current screen mode is returned, it means that the cursor stays outside of the rightmost column.
- (0) is a dummy parameter that can have any value or expression, but generally it is 0.
- In the BHT-7000/BHT-7500, the range of the column numbers does not differ between the normal- and double-width characters.
- For the current row number of the cursor, refer to the CSRLIN function.

Reference:

Statements:	LOCATE and SCREEN
Functions:	CSRLIN

RIGHT String function

RIGHT$

Returns the specified number of rightmost characters from a given string expression.

Syntax:

`RIGHT$(`*`stringexpression`*`,`*`stringlength`*`)`

Parameter:

`stringlength`

A numeric expression which returns a value from 0 to 255.

Description:

Starting at the right side of the string, `RIGHT$` extracts a portion of a string specified by *`stringexpression`* by the number of characters specified by *`stringlength`*.

- If *`stringlength`* is zero, `RIGHT$` returns a null string.
- If *`stringlength`* is greater than the length of *`stringexpression`*, the whole *`stringexpression`* will be returned.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`stringlength`* is out of the range.)

Reference:

Functions: `LEFT$, LEN,` and `MID$`

SEARCH

Searches a specified data file for specified data, and then returns the record number where the search data is found.

Syntax:

```
SEARCH([#]filenumber,fieldvariable,searchdata
[,startrecord])
```

Parameter:

filenumber

A numeric expression which returns a value from 1 to 16.

fieldvariable

A non-array string variable.

searchdata

A string expression.

startrecord

A numeric expression which returns a value from 1 to 32767.

Description:

SEARCH searches a target field specified by *fieldvariable* in a data file specified by *filenumber* for data specified by *searchdata*, starting from a record specified by *startrecord*, and then returns the number of the record where the search data is found.

- *fieldvariable* is a string variable defined by a `FIELD` statement.
- *searchdata* is the data you are looking for.
- *startrecord* is the number of a record where the search is to begin in a data file. The search ends when all of the written records have been searched.

  If you omit *startrecord* option, the search begins at the first record of the data file.
- If the search data is not found, SEARCH returns the value 0.
- A convenient use for SEARCH is, for example, to search for a particular product name, unit price, or stock quantity in a product master file by specifying a bar code data to *searchdata*.
- Since the search begins at a record specified by *startrecord* in a data file and finishes at the last record, sorting records in the data file in the order of frequency of use before execution of this function will increase the searching speed.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range
`34h`	Bad file name or number (You specified *`filenumber`* of an unopened file.)
`36h`	Improper file type (You specified *`filenumber`* of a file other than data files.)
`3Ah`	File number out of the range
`3Eh`	A `PUT` or `GET` statement executed without a `FIELD` statement. (No `FIELD` statement is found.)

Reference:

Statements:	`FIELD`, `GET`, and `OPEN`
Functions:	`LOF`

SOH$

Modifies the value of a header (SOH) for the BHT-protocol; also returns the current value of a header.

Syntax:

Syntax 1 (Changing the value of a header):

`SOH$=`*`stringexpression`*

Syntax 2 (Returning the current value of a header):

`SOH$`

Parameter:

`stringexpression`

A string expression which returns a single-byte character.

Description:

■ Syntax 1

`SOH$` modifies the value of a header (one of the text control characters) which indicates the start of heading text in the BHT-protocol when a data file is transmitted by an `XFILE` statement. (For the BHT-protocol, refer to the BHT User's Manual.)

- `SOH$` is called a protocol function.
- The initial value of a header (SOH) is 01h.

■ Syntax 2

`SOH$` returns the current value of a header.

Run-time errors:

Error code	Meaning
`0Fh`	String length out of the range (*`stringexpression`* is more than a single byte.)

Reference:

Statements:	`OPEN "COM:"` and `XFILE`
Functions:	`ETX$` and `STX$`

STRing String function

STR$

Converts the value of a numeric expression into a string.

Syntax:

`STR$(`*`numericexpression`*`)`

Parameter:

`numericexpression`

A numeric expression.

Description:

STR$ converts the value of *`numericexpression`* into a string.

- If *`numericexpression`* is 0 or positive, STR$ automatically adds a leading space as shown below.

```
PRINT STR$(123);LEN(STR$(123))
```

```
  123  4
```

  To delete the leading space, you should use the `MID$` function as shown below.

```
PRINT MID$(STR$(123),2);LEN(STR$(123))
```

```
123  4
```

- If *`numericexpression`* is negative, STR$ adds a minus sign as shown below.

```
PRINT STR$(-456);LEN(STR$(-456))
```

```
-456  4
```

- A common use for STR$ is to write numeric data into a data file.
- The VAL function has the opposite capability to STR$.

Reference:

Functions: VAL

STX$

Modifies the value of a header (STX) for the BHT-protocol; also returns the current value of a header.

Syntax:

Syntax 1 (Changing the value of a header):

`STX$=`*`stringexpression`*

Syntax 2 (Returning the current value of a header):

`STX$`

Parameter:

`stringexpression`

A string expression which returns a single-byte character.

Description:

■ Syntax 1

STX$ modifies the value of a header (one of the text control characters) which indicates the start of data text in the BHT-protocol when a data file is transmitted by an `XFILE` statement. (For the BHT-protocol, refer to the BHT User's Manual.)

- STX$ is called a protocol function.
- The initial value of a header (STX) is 02h.

■ Syntax 2

STX$ returns the current value of a header.

Run-time errors:

Error code	Meaning
`0Fh`	String length out of the range (*`stringexpression`* is more than a single byte.)

Reference:

Statements:	`OPEN "COM:"` and `XFILE`
Functions:	`ETX$` and `SOH$`

TIME I/O function

TIME$

Returns the current system time or wakeup time, or sets a specified system time or wakeup time.

Syntax:

Syntax 1 (Retrieving the current system time or the wakeup time):

```
TIME$
```

Syntax 2 (Setting the current system time or the wakeup time):

`TIME$="`*`time`*`"`

Parameter:

`time`

A string expression.

Description:

■ Syntax 1

Retrieving the current system time

`TIME$` returns the current system time as an 8-byte string. The string has the format below.

hh:*mm*:*ss*

where *hh* is the hour from 00 to 23 in 24-hour format, *mm* is the minute from 00 to 59, and *ss* is the second from 00 to 59.

Example:

```
CLS
PRINT TIME$
```

Retrieving the wakeup time (For the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500)

`TIME$` returns the wakeup time as a 5-byte string. The string has the format below.

hh:*mm*

■ Syntax 2

Setting the system time

`TIME$` sets the system time specified by `"`*`time`*`."` The format of `"`*`time`*`"` is the same as that in syntax 1.

Example: `TIME$="13:35:45"`

Setting the wakeup time (For the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500)

`TIME$` sets the wakeup time specified by `"`*`time`*`."` The format of `"`*`time`*`"` is the same as that in syntax 1.

- The calendar clock is backed up by the battery. (For the system date, refer to the `DATE$` function.)
- For returning the current wakeup time or setting a specified wakeup time, bit 2 of port 8 should be set to 1 with the `OUT` statement before execution of this function.
- For the wakeup function, refer to Chapter 12, Section 12.3, "Wakeup Function."

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*`time`* is out of the range.)

Reference:

Functions: `DATE$`

TIMER-A/TIMER-B/TIMER-C I/O function

TIMEA/TIMEB/TIMEC

Returns the current value of a specified timer or sets a specified timer.

Syntax:

Syntax 1 (Retrieving the current value of a specified timer):

```
TIMEA
TIMEB
TIMEC
```

Syntax 2 (Setting a specified timer):

```
TIMEA=count
TIMEB=count
TIMEC=count
```

Parameter:

count

A numeric expression which returns a value from 0 to 32767.

Description:

■ Syntax 1

`TIMEA`, `TIMEB`, or `TIMEC` returns the current value of timer-A, -B, or -C, respectively, as a 2-byte integer.

■ Syntax 2

`TIMEA`, `TIMEB`, or `TIMEC` sets the count time specified by *count*.

- *count* is a numeric value in units of 100 ms.
- Upon execution of this function, the Interpreter starts a specified timer counting down in decrements of 100 ms (equivalent to -1) until the timer value becomes 0.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range (*count* is a negative value.)
`06h`	The operation result is out of the allowable range. (*count* is greater than 32767.)

VAL

Converts a string into a numeric value.

Syntax:

`VAL(`*`stringexpression`*`)`

Parameter:

`stringexpression`

A string expression which represents a decimal number.

Description:

VAL converts the string specified by *`stringexpression`* into a numeric value.

- If *`stringexpression`* is nonnumeric, VAL returns the value 0.

```
PRINT VAL("ABC")
```

```
0
```

- If *`stringexpression`* contains a nonnumeric in midstream, VAL converts the string until it reaches the first character that cannot be interpreted as a numeric.

```
PRINT VAL("1.2E-3ABC")
```

```
1.200000000E-03
```

- The STR$ function has the opposite capability to VAL.

Reference:

Functions:	`ASC` and `STR$`

Chapter 16
Extended Functions

CONTENTS

16.1 Overview

In addition to the BHT-BASIC statements and functions, the BHT-7000/BHT-7500/BHT-7500S supports the following extended functions which can be invoked by the `CALL` statement.

Extented functions	Used to:	Integrated in:
`SYSTEM.FN3`	Read or write system settings from/to the memory.	BHT-7000/ BHT-7500/ BHT-7500S
`SS.FN3`	Connect or disconnect the BHT-7500S to/from the spread spectrum system. (For details, see Chapter 17.)	BHT-7500S
`SOCKET.FN3`	Implement a subset of the TCP/IP socket application program interface (API). (For details, see Chapter 18.)	BHT-7500S
`FTP.FN3`	Implement FTP client services for file transfer to/from FTP servers. (For details, see Chapter 18.)	BHT-7500S

16.2 Reading or writing system settings from/to the memory (`SYSTEM.FN3`)

16.2.1 Function Number List of `SYSTEM.FN3`

The `SYSTEM.FN3` may read or write system settings depending upon the function number specified, as listed below.

Function number	Used to:
`0`	Get `SYSTEM.FN3` version
`1`	Read numeric data from System Mode settings
`2`	Write numeric data to System Mode settings
`3`	Read string data from System Mode settings
`4`	Write string data to System Mode settings
`5`	Get font information

16.2.2 Detailed Function Specifications

Function #0: Get `SYSTEM.FN3` version

Syntax: `CALL "SYSTEM.FN3" 0` *`VERSION$`*

Description: This function returns the `SYSTEM.FN3` library version in *`VERSION$`*.

Parameter: (None)

Returned value: *`VERSION$`* Version which is fixed to 7 characters

Function #1: Read numeric data from System Mode settings

Syntax: `CALL "SYSTEM.FN3" 1` *`PARA%,DATA%`*

Description: This function reads numeric data (*`DATA%`*) from the system menu item specified by *`PARA%`*.

Parameter: *`PARA%`* Item number of the system menu

Returned value: *`DATA%`* Numeric data read from the specified system menu item

System menu items list:

Item number (*PARA%*)	System menu item	Attribute[*1]	*DATA%*, numeric data of the system menu item	Initial value
1	Shift key mode	R/W	0: Nonlock 1: Onetime	0
2	Assignment to M1 key	R/W	0: None 1: Enter key 2: Trigger switch 3: Shift key 4: Backlight on/off function key	0
3	Assignment to M2 key	R/W	Same as above.	0
4	Assignment to M3 key	R/W	Same as above.	2
5	Assignment to M4 key	R/W	Same as above.	2
6	Black-and-white inverted label reading function	R/W	0: OFF 1: ON	0
7	Reserved for system	–		
8	Decode level	R/W	1 to 9	4
9	Minimum number of digits to be read for ITF	R/W	2 to 20	4
10	Minimum number of digits to be read for STF	R/W	1 to 20	2

[*1] R/W: Read and write possible
RO: Read only

Item number (*PARA%*)	System menu item	Attribute[*1]	`DATA%`, numeric data of the system menu item	Initial value
`11`	Minimum number of digits to be read for Codabar	R/W	3 to 20	4
`12`	Default interface to be used for user programs	R/W	0: Optical interface 1: Direct-connect interface	0
`13`	Default interface to be used for System Mode	R/W	0: Optical interface 1: Direct-connect interface	0
`14`	Transmission speed for optical interface	R/W	0: 2400 bps 1: 9600 bps 2: 19200 bps 3: 38400 bps 4: 57600 bps 5: 115200 bps	1

`15`
|
`17`
	Reserved for system	–		

`18`	Transmission speed for direct-connect interface	R/W	0: 300 bps 1: 600 bps 2: 1200 bps 3: 2400 bps 4: 4800 bps 5: 9600 bps 6: 19200 bps 7: 38400 bps 8: 57600 bps 9: 115200 bps	6
`19`	Vertical parity for direct-connect interface	R/W	0: None 1: Odd 2: Even	0
`20`	Character length for direct-connect interface	R/W	0: 7 bits 1: 8 bits	1
`21`	Stop bit length for direct-connect interface	R/W	0: 1 bit 1: 2 bits	0
`22`	Serial numbers for optical interface	R/W	0: No numbers (OFF) 1: Add numbers (ON)	1
`23`	Horizontal parity for optical interface	R/W	0: No parity (OFF) 1: Add (ON)	1
`24`	Timeout for data link establishment for optical interface	R/W	0: No timeout 1: 30 sec, 2: 60 sec, 3: 90 sec, 4: 120 sec	1
`25`	Space codes in the tail of a data field for optical interface	R/W	0: Ignore 1: Handle as data	0
`26`	Serial numbers for direct-connect interface	R/W	0: No numbers (OFF) 1: Add numbers (ON)	1
`27`	Horizontal parity for direct-connect interface	R/W	0: No parity (OFF) 1: Add (ON)	1
`28`	Timeout for data link establishment for direct-connect interface	R/W	0: No timeout 1: 30 sec, 2: 60 sec, 3: 90 sec, 4: 120 sec	1
`29`	Space codes in the tail of a data field for direct-connect interface	R/W	0: Ignore 1: Handle as data	0

[*1] R/W: Read and write possible
RO: Read only

Item number (*PARA%*)	System menu item	Attribute[*1]	*DATA%*, numeric data of the system menu item	Initial value
`30`	Communications protocol type	R/W	0: BHT protocol 2: BHT-Ir protocol	0
`31`	Resume function	R/W	0: OFF 1: ON	1[*2]

`32`
|
`34`
	Reserved for system	–		

`35`	RAM size	RO	512/1024/2048 (kilobytes)	[*3]
`36`	ROM size	RO	2048/4096/8192 (kilobytes)	[*3]
`37`	Cluster size	RO	4096 (bytes)	
`38`	Scanning range marker (BHT-7000 only)	R/W	0: Normal mode 1: OFF mode	0

[*1] R/W: Read and write possible
RO: Read only
[*2] The resume function setting made here is effective also in user programs downloaded to the BHT.
[*3] These values will vary depending upon the hardware type.

Function #2: Write numeric data to System Mode settings

Syntax: `CALL "SYSTEM.FN3" 2` *PARA%,DATA%*

Description: This funcion writes numeric data (*DATA%*) to the system menu item specified by *PARA%*.

Parameter: *PARA%* Item number of the system menu
DATA% Numeric data to be specified
(See the system menu items list given in Function #1.)

Returned value: None

System menu items list: Refer to the System menu items list given in Function #1.

Function #3: Read string data from System Mode settings

Syntax: `CALL "SYSTEM.FN3" 3` *`PARA%,DATA$`*

Description: This funcion reads string data (*DATA$*) from the system menu item specified by *PARA%*.

Parameter: *PARA%* Item number of the system menu

Returned value: *DATA$* String data read from the specified system menu item

System menu items list:

Item number (*PARA%*)	System menu item	Attribute	*DATA$*, numeric data of the system menu item
1	System version	RO	"X.XX" fixed to 4 characters
2	Reserved for system	–	
3	Model name	RO	Max. of 8 characters (e.g., "BHT75")
4	Product number assigned to the BHT	RO	Fixed to 16 characters (e.g., "496310….")
5	Serial number assigned to the BHT	R/W	Fixed to 6 characters
6	Execution program	R/W	Filename.xxx (Filename followed by period and extension) If not selected, a null string
7	Version of the BHT system parameter file	RO	"X.XX" fixed to 4 characters

Function #4: Write string data to System Mode settings

Syntax: `CALL "SYSTEM.FN3" 4` *`PARA%,DATA$`*

Description: This funcion writes string data (*DATA$*) to the system menu item specified by *PARA%*.

Parameter: *PARA%* Item number of the system menu
DATA$ String data to be specified
(See the System menu items list given in Function #3.)

Returned value: None

System menu items list: Refer to the System menu items list given in Function #3.

Function #5: Get font information

Syntax: `CALL "SYSTEM.FN3" 5` *`N.FONT%,VERSION$`*`()`

Description: This funcion returns font information--the number of downloaded fonts, font name, font size, and font version.

Parameter: None

Returned value:

`N.FONT%`	Number of fonts
`VERSION$`	Sets of the font name, font size, and font version in the following format

Font name	Font size	Font version
8 bytes	2 bytes	8 bytes

Note: If the number of elements of *`VERSION$`* is less than the number of fonts, the `SYSTEM.FN3` returns the sets of the font information by the number of elements.

Chapter 17
Spread Spectrum Communication (BHT-7500S only)

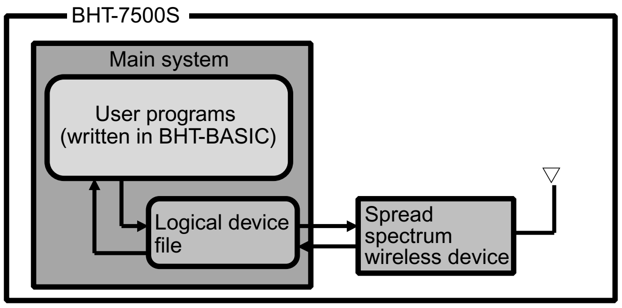
CONTENTS

17.1 Overview

■ Spread spectrum wireless device

The BHT-7500S system consists of the BHT main system and the spread spectrum wireless device; the former executes user programs and the latter performs spread spectrum communications.

User programs use the logical device file (named "COM3") to control the spread spectrum wireless device.

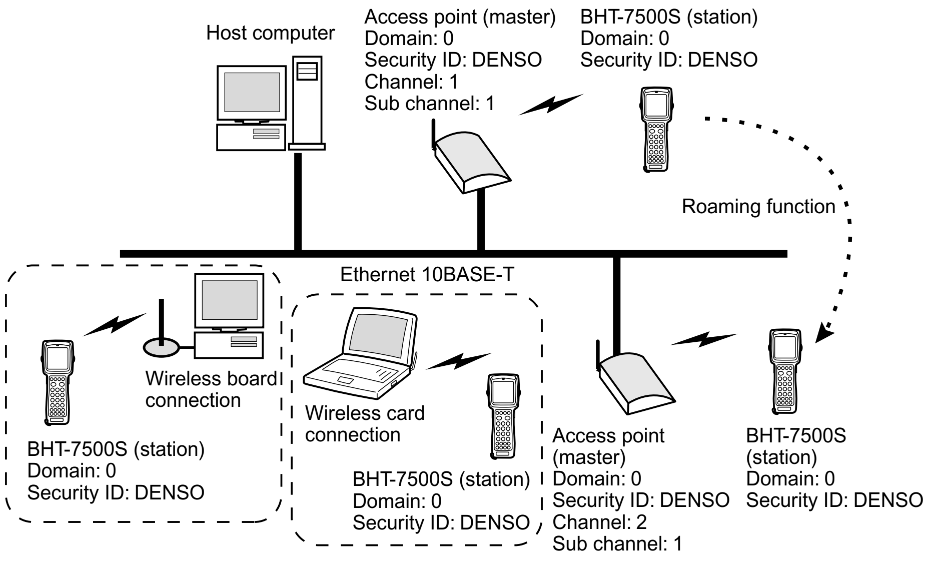

BHT-7500S
Main system
User programs (written in BHT-BASIC)
Logical device file
Spread spectrum wireless device

■ Spread spectrum communications method

The BHT-7500S uses the TCP/IP protocol subset over the spread spectrum wireless device. For details about programming for spread spectrum communications, refer to Chapter 18, "TCP/IP."

■ Configuration of spread spectrum system

Shown below is an example of the spread spectrum system configuration using the BHT-7500S. For details, refer to the BHT-7500/7500S User's Manual.



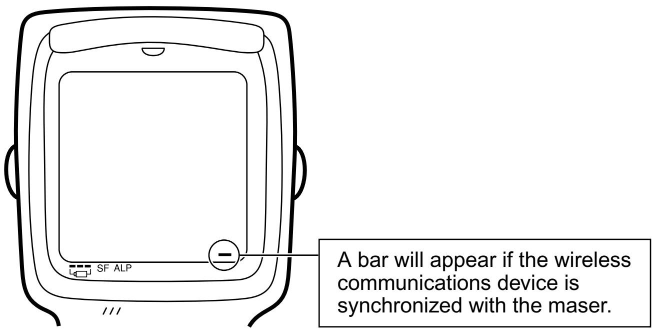
The table below shows the communications status transition as the state of the wireless communications device built in the BHT-7500S changes.

Spread spectrum wireless device	Communication
Open (power on)	Impossible
Checking synchronization with master	Impossible
Synchronization complete	Possible
Roaming	Impossible if roaming leads to the loss of synchronization Possible if synchronization with the master is kept
End of roaming	Possible
Close (power off)	Impossible

If always being opened, the wireless communications device will consume much power. When the device is not in use, therefore, close it as soon as possible.

However, it will take several seconds to open the wireless communications device and synchronize it with the master for making communications ready. Frequent opening and closing of the device will require much time, resulting in slow response. Take into account the application purposes of user programs when programming.

When the wireless communications device is synchronized with the maser, the BHT-7500S will display a bar on the LCD as shown below.


SF ALP
A bar will appear if the wireless communications device is synchronized with the maser.

17.2 Programming for Wireless Communication

When programming for spread spectrum communications, use the following statement and extension functions:

(1) `OPEN` statement (`OPEN "COM3:"`)
Refer to Section 17.3, "Wireless Communications-related Statement."

(2) Spread spectrum library (`SS.FN3`) for controlling the spread spectrum wireless device
Refer to Section 17.4, "Wireless Communication Library (SS.FN3)."

(3) Socket library (`SOCKET.FN3`) for data transmission according to TCP/IP
Refer to Section 18.5, "Socket Library (SOCKET.FN3)."

(4) FTP library (`FTP.FN3`) for file transfer
Refer to Section 18.6, "FTP Library (FTP.FN3)."

17.3 Wireless Communications-related Statement

OPEN "COM3:"Open a wireless communications device file

Syntax: `OPEN "COM3:" AS [#] filenumber`

Description: This statement opens a wireless communications device file.

A wireless communications device file cannot be opened with an optical interface device file concurrently. If you attempt to open them concurrently, a run-time error will occur.

A wireless communications device file can be opened with a bar code device file concurrently.

Syntax error: Refer to Chapter 14, "Statement Reference."

Run-time errors:

Error code	Meaning
`02h`	Syntax error
`37h`	File already open
`3Ah`	File number out of the range
`45h`	File already open (You attempted to open a wireless communications file and the optical interface of a communications device file concurrently.)
`401h`	Failed to open a wireless communications device file.

TIP

A wireless communications device uses TCP/IP for reading or writing data, unlike other communications devices. For details about programming for using TCP/IP over a wireless communications device, refer to Chapter 18, "TCP/IP."

To close a wireless communications device file, use a `CLOSE` statement listed in Chapter 14.

17.4 Wireless Communication Library (SS.FN3)

17.4.1 Overview

The spread spectrum library (SS.FN3) used in a BHT-BASIC CALL statement gets or sets parameters from/to the wireless block.

If wireless communications are frequent, a run-time error may occur when you set or refer to wireless-related parameters. In such a case, set or refer to them again.

■ Function Number List of SS.FN3

Number	Function
0	Get SS.FN3 version
1	Get parameter value (integer) from the wireless block
2	Get parameter value (string) from the wireless block
3	Set parameter value (integer) to the wireless block
4	Set parameter value (string) to the wireless block
7	Check wireless block synchronization with master

17.4.2 Detailed Function Specifications

Function #0 Get SS.FN3 version

Syntax: `CALL "SS.FN3" 0` *`VERSION$`*

Description: This function returns the `SS.FN3` library version in *`VERSION$`*.

Parameters: (None)

Returned value: *`VERSION$`*: Version information, 7 characters, fixed length

Run-time errors:

Error code	Meaning
`F0h`	Mismatch parameter number
`F1h`	Mismatch parameter type
`F2h`	Insufficient string variable storage area

Function #1 Get parameter value (integer) from the wireless block

Syntax: `CALL "SS.FN3" 1` *`PARA%,DATA%`*

Description: This function gets integer (*DATA%*) from the wireless block setting specified by *PARA%*.

Parameters: *PARA%* Setting number

Returned value: *DATA%* Integer read from the specified wireless block setting

Correspondence table:

Setting number (*PARA%*)	Description	Values for setting (*DATA%*)	Initial value
1	Domain information	0 to 15	0

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range
`F0h`	Mismatch parameter number
`F1h`	Mismatch parameter type

Function #2 Get parameter value (string) from the wireless block

Syntax: `CALL "SS.FN3" 2` *`PARA%,DATA$`*

Description: This function gets string (*DATA$*) from the wireless block setting specified by *PARA%*.

Parameters: *PARA%* Setting number

Returned value: *DATA%* String read from the specified wireless block setting

Correspondence table:

Setting number (*PARA%*)	Description	Values for setting (*DATA%*)
1	Wireless block firmware version	Character string, 4 bytes
2	Physical address	Character string, 6 bytes

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range
`34h`	Bad file name or number (The wireless communications device is not opened.)
`F0h`	Mismatch parameter number
`F1h`	Mismatch parameter type
`F2h`	Insufficient string variable storage area
`105h`	Power-off detected
`400h`	Failed to get the setting value. (Failed to set the value.)

NOTE After executing an `OPEN "COM3:"` statement, refer to the above parameter.

Function #3 Set parameter value (integer) to the wireless block

Syntax: `CALL "SS.FN3" 3 PARA%,DATA%`

Description: This function sets integer (*DATA%*) to the wireless block setting specified by *PARA%*.

Parameters: *PARA%* Setting number

Returned value: *DATA%* Integer to be set to the specified wireless block

Correspondence table:

Setting number (*PARA%*)	Description	Values for setting (*DATA%*)
1	Domain information	0 to 15

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range
`37h`	File already open (The wireless communications device has already been opened.)
`F0h`	Mismatch parameter number
`F1h`	Mismatch parameter type

NOTE The above parameter will take effect when the immediately following `OPEN "COM3:"` statement executes.

NOTE The above parameter should be set with the wireless communications device file being closed.

Function #4 Set parameter value (string) to the wireless block

Syntax: `CALL "SS.FN3" 4 PARA%,DATA%`

Description: This function sets string (*DATA$*) to the wireless block setting specified by *PARA%*.

Parameters: *PARA%* Setting number

Returned value: *DATA%* String to be set to the specified wireless block

Correspondence table:

Setting number (*PARA%*)	Description	Values for setting (*DATA%*)
3	Security ID	Character string, maximum 20 bytes

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range
`37h`	File already open (The wireless communications device has already been opened.)
`45h`	Device files prohibited from opening concurrently (The optical interface communications device has been opened.)
`F0h`	Mismatch parameter number
`F1h`	Mismatch parameter type
`105h`	Power-off detected
`400h`	Failed to get the setting value. (Failed to set the value.)

NOTE The above parameter will take effect when the immediately following `OPEN "COM3:"` statement executes.

NOTE Set the above parameter after closing both the optical interface communications device file and wireless communications device file.

NOTE The allowable entry range of the ASCII codes is from 20h to 7Eh. If you set " " to *DATA$*, the default will apply.

Function #7 Check wireless block synchronization with master

Syntax: `CALL "SS.FN3" 7` *`TIMEOUT%,ASSOC%`*

Description: This function checks whether the wireless block is synchronized with the master.

According to the timeout length specified by *`TIMEOUT%`*, the system operates as follows:

- If greater than zero (0) is specified to *`TIMEOUT%`* (recommended), this program will check synchronization with the master during the specified time. Upon completion of synchronization, the program will set zero (0) to *`ASSOC%`* to end the checking operation.

  If the wireless block fails to synchronize with the master within the specified time, the program will set -1 to *`ASSOC%`* to end the checking operation.
- If zero (0) is specified to *`TIMEOUT%`*, this function will check synchronization with the master and immediately return.
- If -1 is specified to *`TIMEOUT%`*, no timeout will occur so that this function will wait until synchronization will be complete.

Parameters: *`TIMEOUT%`* Maximum time (unit: 100ms) to wait for synchronization with master

Returned value: *`ASSOC%`* 0 (Synchronization with master complete)|
-1 (Failed to synchronize with master)

Run-time errors:

Error code	Meaning
`F0h`	Mismatch parameter number
`F1h`	Mismatch parameter type
`105h`	Power-off detected

NOTE After executing an `OPEN "COM3:"` statement, refer to the above parameter.

Chapter 18
TCP/IP

CONTENTS

18.1 Two Sides

18.1.1 BHT-7500S

The BHT-7500S includes two built-in libraries providing BHT-BASIC programs with access to a subset of the TCP/IP family of protocols over the spread spectrum communication system.

`SOCKET.FN3:`	This library implements a subset of the BSD4.4 socket application program interface (API).
`FTP.FN3:`	This library implements FTP client services for file transfers to and from FTP servers.

18.1.2 Hosts

`SOCKET.FN3` and `FTP.FN3` require a host machine with the equivalent TCP/IP functionality and running the appropriate server software.

18.2 TCP/IP over Spread Spectrum System

18.2.1 General Procedure

The following is the procedure for using TCP/IP over a wireless communications device.

[1] Configure Wireless Communications Device

To connect to the wireless communications pathway, specify the following system settings in System Mode or by using the extension library SS.FN3 in a user program:

- Domain
- Security ID

For the procedure in System Mode, refer to the "BHT-7500/BHT-7500S User’s Manual." For the details of the SS.FN3, refer to Section 17.4, "Wireless Communication Library (SS.FN3)" in this manual.

If no system settings are made in a user program, those made in System Mode will apply; if made with SS.FN3, those will become system settings.

Given below is a setting example with SS.FN3:

```
para% = 1                          'Specify domain (#1)
data% = 9                          'Value to be set to domain
call "ss.fn3" 3 para%, data%       'Set domain (SS.FN3 function #3)
para% = 3                          'Specify security ID (#3)
data$ = "9999"                     'Value to be set to security ID
call "ss.fn3" 4 para%, data$       'Set security ID
                                   '(SS.FN3 function #4)
```

[2] Configure TCP/IP System

To connect to the TCP/IP pathway, specify the following system settings in System Mode or by using the extension library SOCKET.FN3 in a user program:

- IP address
- Subnet mask
- Default gateway

These settings will be used in [6].

For the procedure in System Mode, refer to the "BHT-7500/7500S User’s Manual." For the details of the SOCKET.FN3, refer to Section 18.5, "Socket Library (SOCKET.FN3)."

Given below is a setting example with SOCKET.FN3:

```
my.addr$ = "192.168.0.125"                'IP address of the BHT
subnetmask$ = "255.255.255.0"             'Subnet mask
gateway$ = "0.0.0.0"                      'Default gateway
para% = 1                                 'Specify IP address (#1)
call "socket.fn3" 45 para%, my.addr$      'Set IP address
                                          '(SOCKET.FN3 Function #45)
para% = 2                                 'Set subnet mask (#2)
call "socket.fn3" 45 para%, subnetmask$   'Set subnet mask
                                          '(SOCKET.FN3 Function #45)
para% = 3                                 'Specify default gateway (#3)
call "socket.fn3" 45 para%, gateway$      'Set default gateway
                                          '(SOCKET.FN3 Function #45)
```

[3] Declare TCP/IP Communications Pathway

Specify the following system settings by using the socket library (SOCKET.FN3):

- Communications device: Wireless communications device
- Link layer: Ethernet

For the setting procedure with the SOCKET.FN3, refer to Section 18.5, "Socket Library (SOCKET.FN3)."

Given below is a setting example using SOCKET.FN3:

```
iftype% = 2                              'Specify wireless communications device
layermode% = 2                           'Specify Ethernest as a link layer

call "socket.fn3" 40 iftype%, layermode%, interface%
                                         'Specify communications pathway
                                         '(SOCKET.FN3 function #40)
                                         'Returns value in interface%
                                         '(The returned value will be used in
                                         '[6] and [8].)
```

[4] Open Wireless Communications Device

Use the `OPEN "COM3:"` statement.

At the opening time, the following will take place:

- Powering up the wireless block
- Performing the self test of the wireless block
- Initializing the wireless block

For the details, refer to Section 17.3, "Wireless Communications-related Statement."

Given below is an example using the wireless communications-related statement:

```
hCom3% = 1                'Specify a file number to be opened
                          '(The file number will be used also in [9].)

open "COM3:" as #hCom3%   'Open the wireless communications device
                          '(OPEN "COM3:" statement)
```

[5] Check Wireless Communications Device Synchronization with Master

Using a wireless communications device for TCP/IP communication requires synchronizing with the master (e.g., access point). To check the synchronization, use the extension library SS.FN3.

In any of the following cases, a wireless communications device may not be synchronized with the master:

- When a wireless communications device is opened (Opening a wireless communications device and synchronizing with the master will take a few seconds.)
- When a wireless block tries to synchronize with a new master in roaming.
- When a wireless block is moved out of the radio-wave area with the master.
- When a wireless block is moved to a place where there is any radio-wave obstruction between the wireless block and the master.

For details about SS.FN3, refer to Section 17.4, "Wireless Communication Library (SS.FN3)."

Given below is a setting example using SS.FN3.

```
timeout% = 100                        'Set time (10 sec.) to wait for
                                      'synchronization with master.

call "ss.fn3" 7 timeout%, assoc%      'Check synchronization with master.
                                      '(SS.FN3 function #7)
                                      'Returns value in assoc%

if assoc% = -1 then                   'If synchronization is not complete, go
   goto Sync.Erro                     'to Sync.Err.

endif
```

[6] Connect to TCP/IP Communications Pathway

Use the extension library SOCKET.FN3. Connecting to the TCP/IP communications pathway requires the following settings (specified in [2]):

- IP address
- Subnet mask
- Default gateway

There are two ways to specify these parameters.

(a) Use the system settings with the extension library SOCKET.FN3. Refer to Section 18.5, "Socket Library (SOCKET.FN3)."

Given below is an example using SOCKET.FN3.

```
call "socket.fn3" 41 interface%    'Connect to communications pathway
                                   '(SOCKET.FN3 function #41)
                                   'Use the returned value of [3] in
                                   'interface%.
```

(b) Use user-defined values provided by the application with the extension library SOCKET.FN3. Refer to Section 18.5, "Socket Library (SOCKET.FN3)."

Given below is an example using SOCKET.FN3.

```
my.addr$ = "192.168.0.125"         'IP address of the BHT
subnetmask$ = "255.255.255.0"      'Subnet mask
gateway$ = "0.0.0.0"               'Default gateway
call "socket.fn3" 42 interface%, my.addr$, subnetmask$, gateway$
                                   'Connect to communications pathway
                                   '(SOCKET.FN3 function #42)
                                   'Use the returned value of [3] in
                                   'interface%.
```

[7] Transfer Data or File via Socket Interface

To transfer data via the socket interface, use the extension library SOCKET.FN3. Refer to Section 18.3, "Socket API" and Section 18.5, "Socket Library (SOCKET.FN3)."

To transfer file via the socket interface, refer to Section 18.4.3, "Using FTP Client."

[8] Disconnect TCP/IP Communications Pathway

Use the extension library SOCKET.FN3. Refer to Section 18.5, "Socket Library (SOCKET.FN3)."

Given below is an example using SOCKET.FN3.

```
Call "socket.fn3" 43 interface%    'Disconnect TCP/IP communications pathway
                                   '(SOCKET.FN3 function #43)
                                   'Use the returned value of [3] in interface%.
```

[9] Close Spread Spectrum Wireless Device

Use the CLOSE statement in BHT-BASIC.

Closing the device will power off the wireless block. For details about the CLOSE statement, refer to Chapter 14 "Statement Reference."

Given below is an example using the CLOSE statement.

```
close #hCom3%                      'Close the wireless communications device
                                   '(Use CLOSE statement)
                                   'Use the file number specified in [4]
```

For details, refer to the sample programs.

18.2.2 Programming Notes for Socket API According to UDP

The user datagram protocol (UDP) has no flow control, so send/receive data may go missing due to poor line conditions or difference of communications capabilities between wireless and Ethernet. To prevent data missing, be sure to incorporate some flow control process into user programs at both the BHT and host.

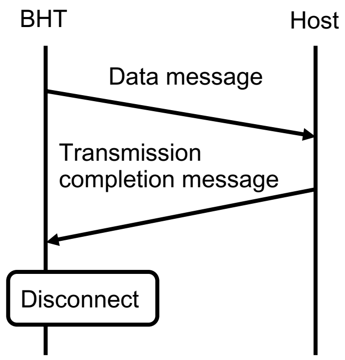
Given below are message transmission examples that support retransmission controls at each of the BHT and host.

■ BHT's retransmission control for a transmission error

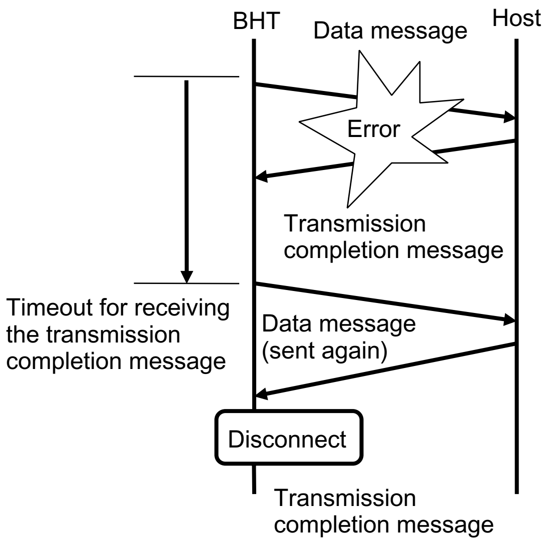
Assume that the BHT uses the protocol of receiving transmission completion message from the host after sending a message.

If the BHT times out for waiting a transmission completion message, it will transmit the unsent message again.

<u>Normal end</u>



<u>Transmission error in a message sent from the BHT</u>

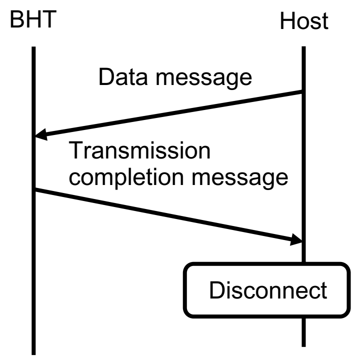

BHT
Data message
Host
Error
Transmission completion message
Timeout for receiving the transmission completion message
Data message (sent again)
Disconnect
Transmission completion message

■ Host’s retransmission control for a transmission error

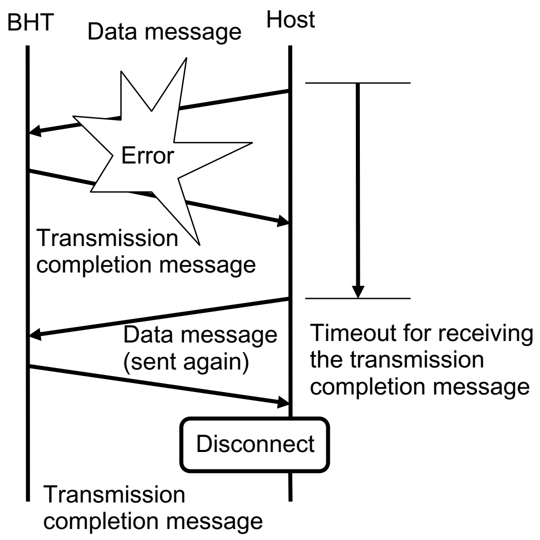
Assume that the host uses the protocol of receiving transmission completion message from the BHT after sending a message.

If the host times out for waiting a transmission completion message, it will transmit the unsent message again.

<u>Normal end</u>



<u>Transmission error in a message sent from the host</u>


BHT
Data message
Host
Error
Transmission completion message
Data message (sent again)
Timeout for receiving the transmission completion message
Disconnect
Transmission completion message

18.2.3 Programming Notes for Resume Function

If the BHT is turned off and on during data transmission in wireless communications, the wireless communications device will remain off so that subsequent data will no longer be sent or received.

In such a case, BHT-BASIC interpreter will return a run-time error (Error code: &h105) informing that the power is off. Develop such user programs that perform the following procedure and then open the wireless communications device again.

■ Procedure for opening the wireless communications device again after detection of a power-off error

(1) Use the `ON ERROR GOTO` statement for error interrupt (at this step, none of (3) through (5) should be carried out)

(2) Use the RESUME statement for transferring control to the main program

(3) Close the socket.

(4) Disconnect the TCP/IP communications pathway.

(5) Close the wireless communications device.

On the next page is a sample program.

```
main:                                         ’Main program
    on error goto Err.TCP                     ’Prepare for error interrupt (To Err.TCP
                                              ’at the time of error occurrence)
                  ⋮
    open "COM3:" as #hCom3%                   ’Open a wireless communications device
                  ⋮                           ’Use the OPEN "COM3:" statement
    sock.stts% = 1                            ’Set "1" to socket processing number
                  ⋮
    call "socket.fn3" 41 interface%           ’Connect TCP/IP communications pathway
                                              ’(system settings)
                  ⋮                           ’Use SOCKET.FN3 function #41
    sock.stts% = 2                            ’Set "2" to socket processing number
                  ⋮
    call "socket.fn3" 26 family%,type%,protocol%,sockfd%     ’Generate socket
                  ⋮                           ’Use SOCKET.FN3 function #26
    sock.stts% = 3                            ’Set "3" to socket processing number
                  ⋮
    call "socket.fn3" 3 sockfd%,family%,port%,serv.addr$     ’Connect socket
                  ⋮                           ’Use SOCKET.FN3 function #3
    sock.stts% = 4                            ’Set "4" to socket processing number
                  ⋮
                  ⋮
    return                                    ’
Err.TCP:                                      ’Error interrupt processing
                                              ’Control transferred to this step if an
                                              ’error occurs
    err.code% = ERR                           ’Get error number
    err.line% = ERL                           ’Get error line number
    resume Sock.Err                           ’RESUME statement to transfer control from
                                              ’error interrupt processing to socket error
                                              ’processing routine
Sock.Err:                                     ’Socket error processing routine
    print " ERR : " ; hex$( err.code% )       ’Display error number
    print " ERL : " ; hex$( err.line% )       ’Display error line number
    if sock.stts% >= 3 then                   ’If OK until socket generation,
      call "socket.fn3" 28 sockfd%            ’close socket
    endif
    if sock.stts% >= 2 then                   ’If OK until connection of TCP/IP
                                              ’communications pathway
      call "socket.fn3" 43 interface%         ’disconnect the pathway
    endif                                     ’
    if sock.stts% >= 1 then                   ’If OK until opening the wireless device
      close #hCom3%                           ’close the device
    endif                                     ’
    goto main                                 ’To main program
```

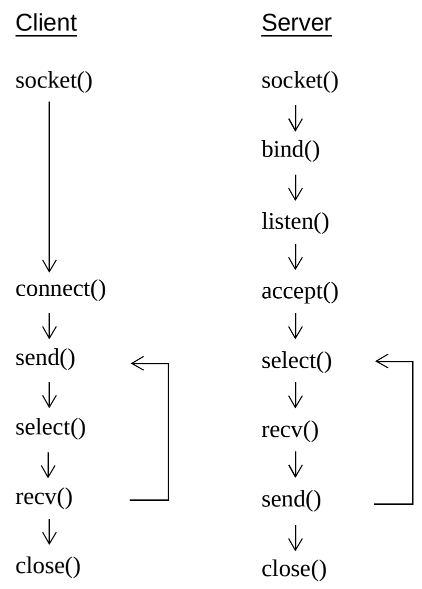
18.3 Socket API

18.3.1 Overview

The `SOCKET.FN3` library implements a subset of the BSD4.4 socket application program interface (API).

The following flowcharts show the BSD4.4 socket API calls for the two communications protocols required for the TCP/IP transport layer: transmission control protocol (TCP) for streams and user datagram protocol (UDP) for datagrams.

■ Transmission Control Protocol (TCP)



■ User Datagram Protocol (UDP)

$$\begin{tikzcd}
\underline{\text{Client}} & \underline{\text{Server}} \\
\text{socket()} \arrow[d] & \text{socket()} \arrow[d] \\
\text{bind()} \arrow[dd] & \text{bind()} \arrow[d] \\
 & \text{select()} \arrow[d] \\
\text{sendto()} \arrow[d] & \text{recvfrom()} \arrow[d] \\
\text{select()} \arrow[d] & \text{sendto()} \arrow[d] \arrow[uu, bend right] \\
\text{recvfrom()} \arrow[d] \arrow[uu, bend right] & \text{close()} \\
\text{close()} &
\end{tikzcd}$$

18.4 FTP Client

18.4.1 Overview

The FTP.FN3 library implements FTP client services for file transfers to and from FTP servers. Note that there are no server capabilities.

This FTP client transfers files between operating systems in image (binary) format. The only translation support is for line delimiter conversion.

In particular, note that this FTP client does not convert between such double-byte character encodings as Shift JIS and EUC. Provide your own code conversion if the server uses a different encoding--for directory and file specifications, in particular.

18.4.2 File Formats

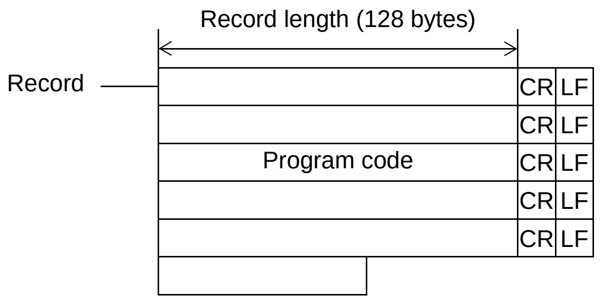
The FTP client classifies files into three types by their extensions: user programs (*.PD3), extension libraries (*.FN3 and *.EX3), and data files (other extensions).

The following describes each file format in turn, assuming that the line delimiter setting specifies the CR-LF combination: a carriage return (0Dh) plus a line feed (0Ah).

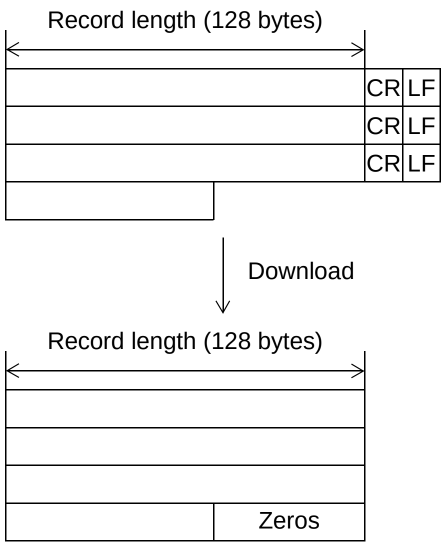
[1] User Programs (*.PD3)

The FTP client reserves the .PD3 extension for user program files generated by the BHT-BASIC compiler.

Program files use a fixed record length of 128 bytes for all records except the last. These records are separated with line delimiters.



The FTP client automatically pads the last record of a downloaded program file with null codes (00h) to maintain the fixed-length format. (The number required is 128 less the number of bytes in the last record).

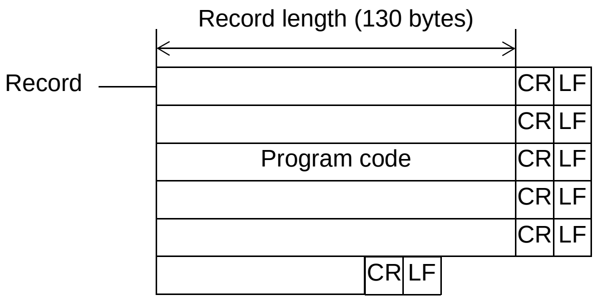


Aside: To conserve memory and boost performance, the BHT packs a pair of ASCII bytes into a single byte by converting each byte into a 4-bit hexadecimal number.

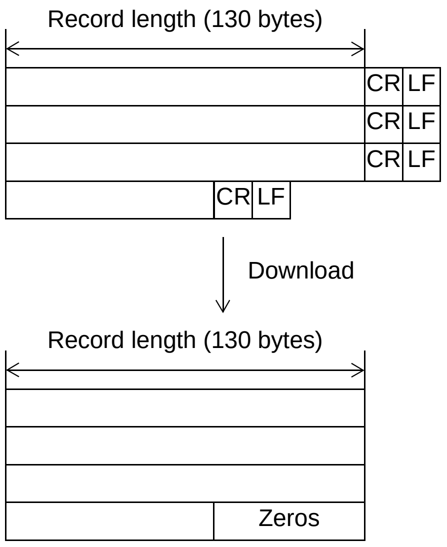
[2] Extension Libraries (*.FN3 and *.EX3)

The FTP client treats files with extensions .FN3 and .EX3 as extension libraries.

Extension libraries use a fixed record length of 130 bytes for all records except the last. These records are separated with line delimiters.



The FTP client automatically pads the last record of a downloaded program file with null codes (00h) to maintain the fixed-length format. (The number required is 130 less the number of bytes in the last record.)



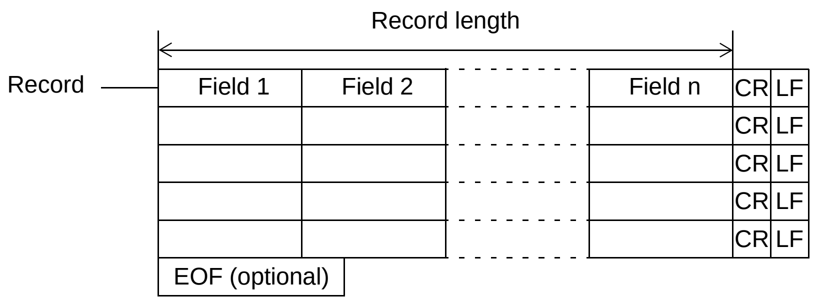
Aside: When downloading extension libraries, the BHT uses 128 bytes out of 130 bytes of record length (the remaining 2 bytes will be used for checking data). To conserve memory and boost performance, the BHT packs a pair of ASCII bytes into a single byte by converting each byte into a 4-bit hexadecimal number.

[3] Data Files

The FTP client treats files with extensions other than .PD3, .FN3, and .EX3 as data files.

Data file records consist of fields separated with line delimiters. An EOF (1Ah) at the end of the data file is optional.

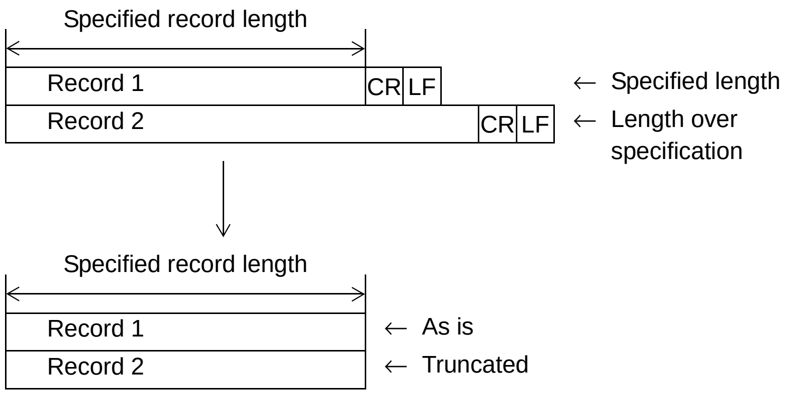
Data files are not limited to ASCII characters. They can use all bytes codes from 00h to FFh.



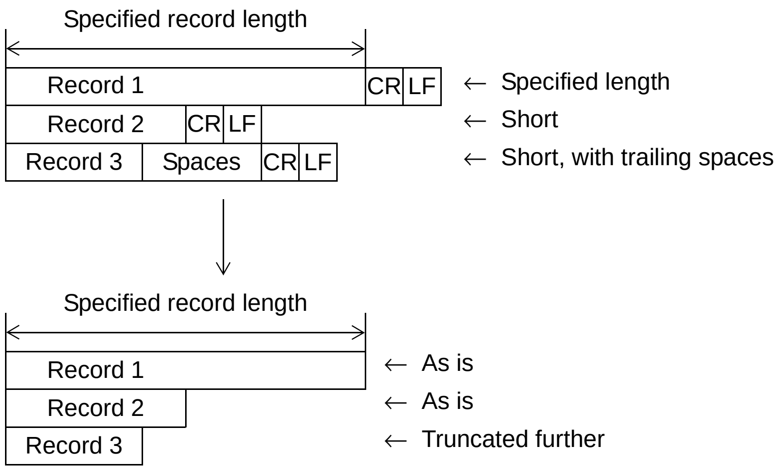
There can be 1 to 16 fields, each 1 to 254 bytes long. The sum of the field lengths and the number of fields, however, must not exceed 255.

If the actual record length is different from the specified record length

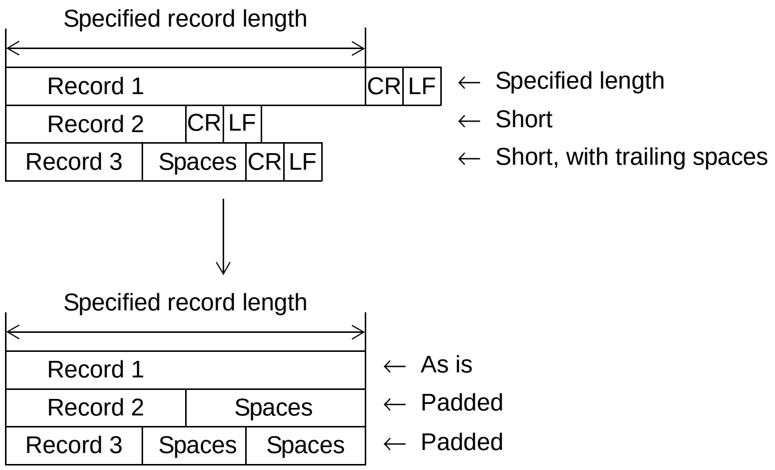
The FTP client discards any excess beyond the specified record length during downloads.



The treatment of short records is under application control. The default is to delete any trailing spaces (20h).

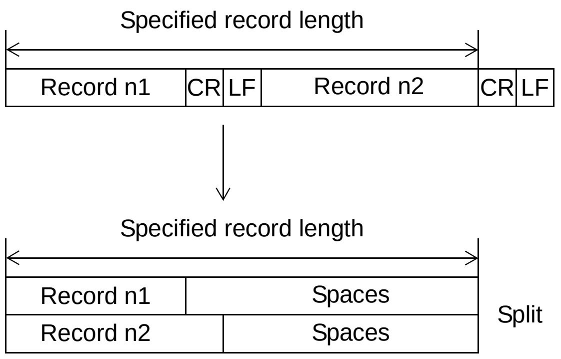


Alternatively, the FTP client can pad such short records to the specified record length with spaces (20h).

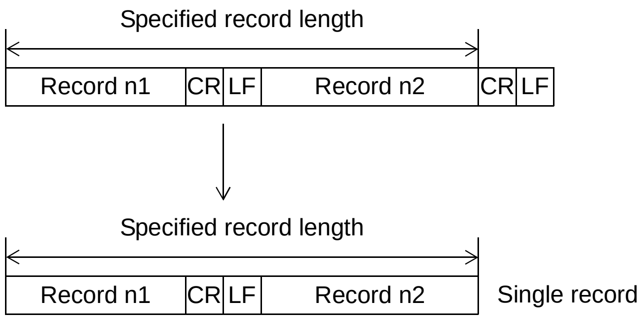


Line Delimiters inside Data Records

The treatment of line delimiters (CR-LF, CR, or LF) inside downloaded data records, which can use all codes from 00h to FFh, is under application control. The default, described above, is to split the incoming stream into short records.



Alternatively, the FTP client can ignore any line delimiters inside downloaded data records, treating them as data. Note, however, that the specified line delimiters must appear in the specified positions between records. Otherwise, the FTP client cancels the transfer with an error because a record is either too long or too short.



18.4.3 Using FTP Client

[1] Basic Procedure

First, set up for using the FTP client, as necessary, with the following steps. All three are optional, but the last two are highly recommended for downloads.

(1) Configure the FTP client with the appropriate `FTP.FN3` extension functions.

(2) Use the `FRE` function to check whether there is sufficient free memory available to hold the downloaded file.

(3) Use a BHT-BASIC `OUT` statement to optimize the drive.

The rest of the procedure is the same as in Section 18.2, "TCP/IP over Spread Spectrum System." The key step is to use the `FTP.FN3` extension functions for the file transfers.

[2] Configuring FTP Client

The FTP client requires the following information before it can transfer files.

- IP address for server
- Login (user) name for server
- Password for that login (user) name

`SOCKET.FN3` provides functions #8 and #9 for reading and changing these settings. For further details on these two functions, see their descriptions in Section 18.6, "FTP Library (FTP.FN3)," Subsection 18.6.2.

[3] Calculating Memory Requirements

The FTP protocol specifications do not provide for checking the amount of BHT memory available during downloads. If the BHT runs out of memory during a download, the FTP client cancels the transfer and deletes the partially downloaded file. The user application program must, therefore, check availability with the FRE function or equivalent method and compare the result with the BHT file size (BFS) before using the download function. The formula for calculating the BHT memory requirements (MEM) depends on the file format.

NOTE

* The line delimiter size (LDS) refers to the number of bytes in each line delimiter: two for operating systems using the CR-LF combination and one for those using only LF or CR.

* The number 4096 (4K) is the assumed memory management unit. Change this to 8192 (8K) if the handy terminal uses that larger block size.

* HFS = host file size

■ User Programs (*.PD3)

Determine MEM from HFS.

BFS = ROUND_UP (HFS $\div$ (128 + LDS)) $\times$ 64

MEM = ROUND_UP (BFS $\div$ 4096) $\times$ 4096

Example: File size of 12,345 bytes on operating system using CR-LF combination

BFS = ROUND_UP (12345 $\div$ (128 + 2)) $\times$ 64 = ROUND_UP(94.96) $\times$ 64 = 6080

MEM = ROUND_UP (6080 $\div$ 4096) $\times$ 4096 = ROUND_UP(1.48) $\times$ 4096 = 8192

Note that 128K of free memory is enough to download even the largest (128K) BASIC program.

■ Extension Libraries (*.FN3 and *.EX3)

Determine MEM from HFS.

BFS = ROUND_UP (HFS $\div$ (130 + LDS)) $\times$ 64

MEM = ROUND_UP (BFS $\div$ 4096) $\times$ 4096

The rest of the procedure is the same as for BASIC program files.

■ Data Files

Determine MEM from the field lengths and number of records.

BPR = bytes per record = (number of fields) + (sum of field lengths)

RPB = records per block = ROUND_DOWN (4096 ÷ BPR)

MEM = ROUND_UP (records ÷ RPB) × 4096

Example: File with 1000 records with four fields of lengths 13, 12, 6, and 1

BPR = 4 + (13 + 12 + 6 + 1) = 36

RPB = ROUND_UP (4096 ÷ 36) = ROUND_UP (113.778) = 113

MEM = ROUND_UP (1000 ÷ 113) × 4096 = ROUND_UP (8.850) × 4096
= 9 × 4096 = 36,864

[4] Optimizing Drive (Recommended)

File system delays can sometimes retard file FTP downloads. The surest way to prevent such delays is to use a BHT-BASIC `OUT` statement to optimize the drive.

Another reason for recommending this step is that it reduces air time, the period that the wireless device is open.

[5] FTP Transfers

The following is the basic procedure for transferring files with the `FTP.FN3` extension functions.

(1) Open an FTP client session with function #1 or #2.

(2) Verify the FTP server current directory with function #4 or #5, if necessary.

(3) Download and upload files with functions #6 and #7.

(4) Close the FTP client session with function #3.

18.5 Socket Library (`SOCKET.FN3`)

18.5.1 Overview

■ String Variables

The following are the string variables used by this library together with their memory requirements.

Description	Name	Size in Bytes
Version information	`VERSION$`	min. 7
Internet address	`IPADDRESS$`	min. 15
Subnet mask	`SUBNETMASK$`	min. 15
Default gateway	`GATEWAY$`	min. 15
Receive buffer	`RECVBUFF$`	1 to 255
Transmit buffer	`SENDBUFF$`	1 to 255

Socket identifier set	`SOCKFDSET$`
	`READFDSET$`
	`WRITEFDSET$`
	`EXCEPTFDSET$`
		min. 41
		min. 41
		min. 41
		min. 41

■ String Array Variables

The following are the string array variables used by this library together with their memory requirements.

Description	Name	Size in Bytes
Receive buffer	`RECVBUFF$()`	1 to 4096
Transmit buffer	`SENDBUFF$()`	
TCP		1 to 4096
UDP		1 to 1472

■ Function Number List

Number	Function	Corresponding Socket API Function
0	Get socket.FN3 version	—
1*	—	accept()
2	Assign address to socket	bind()
3	Connect socket	connect()
4*	—	getpeername()
5*	—	getsockname()
6	Get socket option	getsockopt()
7	Convert host long (4 bytes) to network byte order	htonl()
8	Convert host short (2 bytes) to network byte order	htons()
9	Convert Internet address from dotted quad notation to 32-bit integer	inet_addr()
10*	—	listen()
11	Convert network long (4 bytes) to host byte order	ntohl()
12	Convert network short (2 bytes) to host byte order	ntohs()
13*	—	readv()
14	Receive message from TCP socket	recv()
15	Receive message from UDP socket	recvfrom()
16*	—	rresvport()
17	Monitor socket requests	select()
18	Initialize socket identifier set	FD_ZERO macro
19	Add socket identifier to socket identifier set	FD_SET macro
20	Delete socket identifier from socket identifier set	FD_CLR macro
21	Get socket identifier status from socket identifier set	FD_ISSET macro
22	Send message to another TCP socket	send()
23	Send message to another UDP socket	sendto()
24	Set socket options	setsockopt()
25	Shut down socket	shutdown()
26	Create socket	socket()
27*	—	writev()
28	Close socket	close()

* Socket API function not supported by SOCKET.FN3 library.

Number	Function	Corresponding Socket API Function
40	Specify TCP/IP communications pathway	Unique to BHT
41	Connect TCP/IP communications pathway with system settings	Unique to BHT
42	Connect TCP/IP communications pathway with user settings	Unique to BHT
43	Disconnect TCP/IP communications pathway	Unique to BHT
44	Get TCP/IP system settings	Unique to BHT
45	Set TCP/IP system settings	Unique to BHT
46	Get TCP socket status	Unique to BHT

* Socket API function not supported by SOCKET.FN3 library.

18.5.2 Detailed Function Specifications

Function #0 Get SOCKET.FN3 version

Syntax:	`CALL "SOCKET.FN3" 0 VERSION$`
Description:	This function returns the SOCKET.FN3 library version in `VERSION$`.
Parameters:	(None)
Return value:	`VERSION$:` Version information, 7 characters, fixed length

Function #2 Assign address to socket

Syntax: `CALL "SOCKET.FN3" 2 SOCKFD%, FAMILY%, PORT%,` *`address`*

where *`address`* is ADDRESS or `IPADDRESS$`

Description: This function assigns an address to the specified socket identifier.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API bind() function.

Parameters:

`SOCKFD%`	Socket identifier
`FAMILY%`	Protocol family
`PORT%`	Port
`ADDRESS`	Local address for connection
`IPADDRESS$`	Internet address in dotted quad notation

The protocol family (`FAMILY%`) must be 2, the value indicating the ARPA Internet protocols.

Return value: (None)

Run-time errors:

Error code	Meaning
`209h`	Socket identifier is invalid.
`216h`	A parameter is invalid, or the socket is already bound.
`224h`	The socket is being assigned an address.
`230h`	The specified IP address is already in use.

Function #3: Connect socket

Syntax: `CALL "SOCKET.FN3" 3 SOCKFD%, FAMILY%, PORT%,` *`address`*

where *`address`* is `ADDRESS` or `IPADDRESS$`

Description: This function connects the specified socket identifier to another socket.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API connect() function.

Parameters:

`SOCKFD%`	Socket identifier
`FAMILY%`	Protocol family
`PORT%`	Port
`ADDRESS`	Local address for connection
`IPADDRESS$`	Internet address in dotted quad notation

The protocol family (`FAMILY%`) must be 2, the value indicating the ARPA Internet protocols.

Return value: (None)

Run-time errors:

Error code	Meaning
`105h`	Power-off detected. (BHT-7500S only)
`201h`	Cannot connect to socket.
`209h`	Socket identifier is invalid.
`216h`	A parameter is invalid.
`229h`	The specified socket does not match the connection target socket.
`22Fh`	The specified address family is invalid for this socket.
`230h`	The specified address is already in use.
`231h`	The specified address is invalid.
`238h`	The specified socket is already connected.
`23Ch`	The connection attempt has timed out.
`23Dh`	Failed to connect.
`241h`	There is no connection pathway to the host for TCP socket.

Function #6: Get socket option

Syntax:	`CALL "SOCKET.FN3" 6 SOCKFD%, OPTNAME%, ` *`option`*
	where *`option`* is `OPTION%` or `OPTION`
Description:	This function gets the specified option setting for the specified socket.
	BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API getsockopt() function.

Parameters:

`SOCKFD%`	Socket identifier
`OPTNAME%`	Option name

Return value:

`option`	Current setting for socket option (`OPTION%`/`OPTION`) of type integer/real

Correspondence tables:

Option Number (`OPTNAME%`)	Description	Values for Option (`OPTION%`)
2	Keep-alive timer enable/disable	0 (disabled), 1 (enabled)

Option Number (`OPTNAME%`)	Description	Values for Option (`OPTION`)
8	Transmit buffer size (byte)	1 to 8192
9	Receive buffer size (byte)	1 to 8192
26	Retry count	0 to 32
30	Initial round trip time (ms)	100 to 3000
31	Minimum round trip time (ms)	100 to 1000
32	Maximum round trip time (ms)	100 to 60000

Run-time errors:

Error code	Meaning
`209h`	Socket identifier is invalid.
`216h`	A parameter is invalid.

Function #7: Convert host long (4 bytes) to network byte order

Syntax: `CALL "SOCKET.FN3" 7 HOSTLONG, NETLONG`

Description: This function converts a (4-byte) long from host byte order to network byte order.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API htonl() function.

Parameters:	`HOSTLONG`	Long in host byte order
Return value:	`NETLONG`	Long in network byte order

Function #8: Convert host short (2 bytes) to network byte order

Syntax: `CALL "SOCKET.FN3" 8 HOSTSHORT%, NETSHORT%`

Description: This function converts a (2-byte) short from host byte order to network byte order.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API htons() function.

Parameters:	`HOSTSHORT%`	Short in host byte order
Return value:	`NETSHORT%`	Short in network byte order

Function #9: Convert Internet address from dotted quad notation to 32-bit integer

Syntax: `CALL "SOCKET.FN3" 9 IPADDRESS$, ADDRESS`

Description: This function converts an Internet address in dotted quad notation to a 4-byte Internet address.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API inet_addr() function.

Parameters:	`IPADDRESS$`	Internet address in dotted quad notation
Return value:	`ADDRESS`	4-byte Internet address

Function #11: Convert network long (4 bytes) to host byte order

Syntax: `CALL "SOCKET.FN3" 11 NETLONG, HOSTLONG`

Description: This function converts a (4-byte) long from network byte to host byte order.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API ntohl() function.

Parameters:	`NETLONG`	Long in network byte order
Return value:	`HOSTLONG`	Long in host byte order

Function #12: Convert network short (2 bytes) to host byte order

Syntax: `CALL "SOCKET.FN3" 12 NETSHORT%, HOSTSHORT%`

Description: This function converts a (2-byte) short from network byte order to host byte order.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API ntohs() function.

Parameters:	`NETSHORT%`	Short in network byte order
Return value:	`HOSTSHORT%`	Short in host byte order

Function #14: Receive message from TCP socket

Syntax: `CALL "SOCKET.FN3" 14 SOCKFD%, RECVBUFF$[()], RECVLEN%, RECVMODE%, RECVSIZE% [,RECVFLAG%]`

Description: This function receives data from the IP address and port number connected to the specified socket identifier into the specified buffer.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API recv() function.

Parameters:

`SOCKFD%`	Socket identifier
`RECVBUFF$[()]`	Receive buffer
`RECVLEN%`	Maximum number of bytes to receive
`RECVMODE%`	Receive mode
`RECVFLAG%`	Storage method (optional)

The receive buffer (`RECVBUFF$`) can be either a string or string array variable. The maximum size for a string is 255 bytes; for a string array, 4096.

The receive mode (`RECVMODE%`) must be one of the following values:

0	Normal
1	Out of band data
2	Peek at next message

The storage method (`RECVFLAG%`) is required for a string array buffer. It is ignored for a string variable and new data will be written.

The storage method (`RECVFLAG%`) must be one of the following values:

0	Append data to buffer (default if omitted)
1	Overwrite buffer with data

Note: If `RECVFLAG%` is 0 or omitted, the user application program must initialize the receive buffer string array variable before receiving any data.

Return value: `RECVSIZE%` Number of bytes received

Run-time errors:

Error code	Meaning
`105h`	Power-off detected. (BHT-7500S only)
`209h`	Socket identifier is invalid.
`216h`	A parameter is invalid.
`228h`	The maximum number of bytes to receive is too small.
`236h`	An RST from the opposite end has forced connection.
`237h`	There is insufficient system area memory.
`239h`	The specified socket is not connected.
`23Ah`	The specified TCP socket has been closed.

Example: Append operation

Incoming data: 1024 bytes ("0123456789..........0123")

Receive buffer: 8 elements, 128 characters each for a total of 1024 bytes

- After initializing receive buffer

	1	2	3	4	5	[Strings]	125	126	127	128
Element 0	–	–	–	–	–	· · · · · · · · ·	–	–	–	–
Element 7	–	–	–	–	–	· · · · · · · · ·	–	–	–	–

- After receiving first 512 bytes

	1	2	3	4	5	[Strings]	125	126	127	128
Element 0	‘0’	‘1’	‘2’	‘3’	‘4’	· · · · · · · · ·	‘4’	‘5’	‘6’	‘7’
Element 1	‘8’	‘9’	‘0’	‘1’	‘2’	· · · · · · · · ·	‘2’	‘3’	‘4’	‘5’
Element 3	‘4’	‘5’	‘6’	‘7’	‘8’	· · · · · · · · ·	‘8’	‘9’	‘0’	‘1’
Element 4	–	–	–	–	–	· · · · · · · · ·	–	–	–	–
Element 7	–	–	–	–	–	· · · · · · · · ·	–	–	–	–

- After receiving remaining 512 bytes

	1	2	3	4	5	[Strings]	125	126	127	128
Element 0	‘0’	‘1’	‘2’	‘3’	‘4’	· · · · · · · · ·	‘4’	‘5’	‘6’	‘7’
Element 1	‘8’	‘9’	‘0’	‘1’	‘2’	· · · · · · · · ·	‘2’	‘3’	‘4’	‘5’
Element 3	‘4’	‘5’	‘6’	‘7’	‘8’	· · · · · · · · ·	‘8’	‘9’	‘0’	‘1’
Element 4	‘2’	‘3’	‘4’	‘5’	‘6’	· · · · · · · · ·	‘6’	‘7’	‘8’	‘9’
Element 7	‘6’	‘7’	‘8’	‘9’	‘0’	· · · · · · · · ·	‘0’	‘1’	‘2’	‘3’

Second half is appended to first.

Example: Overwrite operation

Incoming data: 1024 bytes ("0123456789..........0123")

Receive buffer: 8 elements, 128 characters each for a total of 1024 bytes

- After initializing receive buffer

[Strings]

	1	2	3	4	5		125	126	127	128
Element 0	–	–	–	–	–	· · · · · · · · ·	–	–	–	–
Element 7	–	–	–	–	–	· · · · · · · · ·	–	–	–	–

- After receiving first 512 bytes

[Strings]

	1	2	3	4	5		125	126	127	128
Element 0	‘0’	‘1’	‘2’	‘3’	‘4’	· · · · · · · · ·	‘4’	‘5’	‘6’	‘7’
Element 1	‘8’	‘9’	‘0’	‘1’	‘2’	· · · · · · · · ·	‘2’	‘3’	‘4’	‘5’
Element 3	‘4’	‘5’	‘6’	‘7’	‘8’	· · · · · · · · ·	‘8’	‘9’	‘0’	‘1’
Element 4	–	–	–	–	–	· · · · · · · · ·	–	–	–	–
Element 7	–	–	–	–	–	· · · · · · · · ·	–	–	–	–

- After receiving remaining 512 bytes

[Strings]

	1	2	3	4	5		125	126	127	128
Element 0	‘2’	‘3’	‘4’	‘5’	‘6’	· · · · · · · · ·	‘6’	‘7’	‘8’	‘9’
Element 1	‘0’	1’	‘2’	‘3’	‘4’	· · · · · · · · ·	‘4’	‘5’	‘6’	‘7’
Element 3	‘6’	‘7’	‘8’	‘9’	‘0’	· · · · · · · · ·	‘0’	‘1’	‘2’	‘3’
Element 4	–	–	–	–	–	· · · · · · · · ·	–	–	–	–
Element 7	–	–	–	–	–	· · · · · · · · ·	–	–	–	–

Second half overwrites first.

Function #15: Receive message from UDP socket

Syntax: `CALL "SOCKET.FN3" 15 SOCKFD%, RECVBUFF$[()], RECVLEN%, RECVMODE%, FAMILY%, PORT%, address, RECVSIZE% [,RECVFLAG%]`

where `address` is `ADDRESS` or `IPADDRESS$`

Description: This function receives data from the IP address and port number connected to the specified socket identifier into the specified buffer.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API recvfrom() function.

Parameters:

`SOCKFD%`	Socket identifier
`RECVBUFF$[()]`	Receive buffer
`RECVLEN%`	Maximum number of bytes to receive
`RECVMODE%`	Receive mode
`FAMILY%`	Protocol family
`PORT%`	Port
`ADDRESS`	Local address for connection
`IPADDRESS$`	Internet address in dotted quad notation
`RECVFLAG%`	Storage method (optional)

The receive buffer (`RECVBUFF$`) can be either a string or string array variable. The maximum size for a string is 255 bytes; for a string array, 4096.

The receive mode (`RECVMODE%`) must be one of the following values:

0	Normal
1	Out of band data
2	Peek at next message

The protocol family (`FAMILY%`) must be 2, the value indicating the ARPA Internet protocols.

The storage method (`RECVFLAG%`) is required for a string array buffer. It is ignored for a string variable and new data will be written.

The storage method (`RECVFLAG%`) must be one of the following values:

0	Append data to buffer (default if omitted)
1	Overwrite buffer with data

Note: If `RECVFLAG%` is 0 or omitted, the user application program must initialize the receive buffer string array variable before receiving any data.

Return value:

`RECVSIZE%`	Number of bytes received

Run-time errors:

Error code	Meaning
105h	Power-off detected. (BHT-7500S only)
209h	Socket identifier is invalid.
216h	A parameter is invalid.
228h	The maximum number of bytes to receive is too small.
229h	TCP is the wrong protocol here.
237h	There is insufficient system area memory.

Function #17: Monitor socket requests

Syntax: `CALL "SOCKET.FN3" 17 MAXFD%, READFDSET$, WRITEFD-SET$, EXCEPTFDSET$, TIMEOUT, RESULT%`

Description: This function waits for changes in the socket identifier sets (read, write, and exception conditions) for the specified socket identifiers.

The only exception condition is out of band data.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API select() function.

Parameters:

`MAXFD%`	Number of socket identifiers + 1
`READFDSET$`	Socket identifier set to monitor for read
`WRITEFDSET$`	Socket identifier set to monitor for write
`EXCEPTFDSET$`	Socket identifier set to check for exception conditions
`TIMEOUT`	Waiting period (in seconds)

The waiting period (`TIMEOUT`) must be one of the following values:

-1	No waiting period
0	No timeout

Other time interval in seconds

Return value: `RESULT%` Number of sockets that are ready.
After a timeout, `RESULT%` contains 0.

Run-time errors:

Error code	Meaning
`105h`	Power-off detected. (BHT-7500S only)
`209h`	Socket identifier is invalid.
`216h`	A parameter is invalid.

Function #18: Initialize socket identifier set

Syntax: `CALL "SOCKET.FN3" 18 SOCKFDSET$`

Description: This function initializes the specified socket identifier set.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API FD_ZERO macro.

Parameters: `SOCKFDSET$` Socket identifier set

Return value: (None)

Function #19: Add socket identifier to socket identifier set

Syntax: `CALL "SOCKET.FN3" 19 SOCKFD%, SOCKFDSET$`

Description: This function adds the specified socket identifier to the specified identifier set.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API FD_SET macro.

Parameters:

`SOCKFD%`	Socket identifier
`SOCKFDSET$`	Socket identifier set

Return value: (None)

Function #20: Delete socket identifier from socket identifier set

Syntax: `CALL "SOCKET.FN3" 20 SOCKFD%, SOCKFDSET$`

Description: This function deletes the specified socket identifier from the specified identifier set.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API FD_CLR macro.

Parameters:

`SOCKFD%`	Socket identifier
`SOCKFDSET$`	Socket identifier set

Return value: (None)

Function #21: Get socket identifier status from socket identifier set

Syntax: `CALL "SOCKET.FN3" 21 SOCKFD%, SOCKFDSET$, FDISSET%`

Description: This function gets the status of the specified socket identifier in the specified socket identifier set.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API FD_ISSET macro.

Parameters:

`SOCKFD%`	Socket identifier
`SOCKFDSET$`	Socket identifier set

Return value:

`FDISSET%`	Socket identifier status

The socket identifier status (FDISSET%) has the following values:

0	No change
1	Change in status

Function #22: Send message to another TCP socket

Syntax: `CALL "SOCKET.FN3" 22 SOCKFD%, SENDBUFF$[()], SENDLEN%, SENDMODE%, SENDSIZE%`

Description: This function transmits data from the specified buffer to the IP address and port number connected to the specified socket identifier.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API send() function.

Parameters:

`SOCKFD%`	Socket identifier
`SENDBUFF$[()]`	Transmit buffer
`SENDLEN%`	Number of bytes to transmit
`SENDMODE%`	Transmit mode

The transmit buffer (`SENDBUFF$`) can be either a string or string array variable. The maximum size for a string is 255 bytes; for a string array, 4096.

The transmit mode (`SENDMODE%`) must be one of the following values:

0	Normal
1	Out of band data
4	Bypass pathway control function

Return value: `SENDSIZE%` Number of bytes transmitted

Run-time errors:

Error code	Meaning
`105h`	Power-off detected. (BHT-7500S only)
`209h`	Socket identifier is invalid.
`228h`	The maximum number of bytes to receive is too small.
`237h`	There is insufficient system area memory.
`239h`	The specified socket is not connected.
`23Ah`	The specified TCP socket has been closed.
`241h`	There is no connection pathway to the host for UDP socket.

Function #23: Send message to another UDP socket

Syntax: `CALL "SOCKET.FN3" 23 SOCKFD%, SENDBUFF$[()], SENDLEN%, SENDMODE%, FAMILY%, PORT%, address, SENDSIZE%`

where `address` is `ADDRESS` or `IPADDRESS$`

Description: This function transmits data from the specified buffer to the IP address and port number connected to the specified socket identifier.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API sendto() function.

Parameters:

`SOCKFD%`	Socket identifier
`SENDBUFF$[()]`	Transmit buffer
`SENDLEN%`	Number of bytes to transmit
`SENDMODE%`	Transmit mode
`FAMILY%`	Protocol family
`PORT%`	Port
`ADDRESS`	Local address for connection
`IPADDRESS$`	Internet address in dotted quad notation

The transmit buffer (`SENDBUFF$`) can be either a string or string array variable. The maximum size for a string is 255 bytes; for a string array, 1472.

The transmit mode (`SENDMODE%`) must be one of the following values:

0	Normal
1	Out of band data
4	Bypass pathway control function

The protocol family (`FAMILY%`) must be 2, the value indicating the ARPA Internet protocols.

Return value: `SENDSIZE%` Number of bytes transmitted

Run-time errors:

Error code	Meaning
`105h`	Power-off detected. (BHT-7500S only)
`209h`	Socket identifier is invalid.
`228h`	The maximum number of bytes to receive is too small.
`229h`	TCP is the wrong protocol here.
`237h`	There is insufficient system area memory.
`241h`	There is no connection pathway to the host.

Function #24: Set socket options

Syntax: `CALL "SOCKET.FN3" 24 SOCKFD%, OPTNAME%, option`

where `option` is `OPTION%` or `OPTION`

Description: This function sets the specified option for the specified socket to the new value.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API setsockopt() function.

Parameters:

`SOCKFD%`	Socket identifier
`OPTNAME%`	Option name
`OPTION%/OPTION`	New setting for socket option of type integer/real

Return value: (None)

Correspondence tables:

Option Number (`OPTNAME%`)	Description	Values for Option (`OPTION%`)	Initial values
2	Keep-alive timer enable/disable	0 (disabled), 1 (enabled)	0

Option Number (`OPTNAME%`)	Description	Values for Option (`OPTION`)	Initial values
8	Transmit buffer size (byte)	1 to 8192	8192
9	Receive buffer size (byte)	1 to 8192	8192
26	Retry count	0 to 32	12
30	Initial round trip time (ms*)	100 to 3000	3000
31	Minimum round trip time (ms*)	100 to 1000	100
32	Maximum round trip time (ms*)	100 to 60000	60000

* To be set in units of 100.

Run-time errors:

Error code	Meaning
`201h`	Cannot set option after connection established.
`209h`	Socket identifier is invalid.
`216h`	A parameter is invalid.

Function #25: Shut down socket

Syntax: `CALL "SOCKET.FN3" 25 SOCKFD%, HOWTO%`

Description: This function shuts down socket transfers in the specified direction.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API shutdown() function.

Parameters:

`SOCKFD%`	Socket identifier
`HOWTO%`	Direction specification

The direction specification (`HOWTO%`) must be one of the following values:

0	Receive
1	Transmit
2	Both

Return value: (None)

Run-time errors:

Error code	Meaning
`105h`	Power-off detected. (BHT-7500S only)
`209h`	Socket identifier is invalid.
`216h`	A parameter is invalid.
`22Ah`	This option is not recognized at the specification level.

Function #26: Create socket

Syntax: `CALL "SOCKET.FN3" 26  FAMILY%, TYPE%, PROTOCOL%, SOCKFD%`

Description: This function creates a socket from the specified protocol family, socket type, and protocol layer and assigns it to a socket identifier.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API socket() function.

Parameters:

`FAMILY%`	Protocol family for the socket
`TYPE%:`	Socket type
`PROTOCOL%`	Protocol layer for the socket

The protocol family (`FAMILY%`) must be 2, the value indicating the ARPA Internet protocols.

The socket type (`TYPE%`) must be one of the following values:

1	Stream socket
2	Datagram socket
3	Raw socket

The protocol layer (`PROTOCOL%`) must be one of the following values:

1	ICMP
6	TCP
17	UDP

Return value: `SOCKFD%` Socket identifier

Run-time errors:

Error code	Meaning
`218h`	Too many sockets.
`22Bh`	This protocol family does not support the specified protocol type and protocol.
`237h`	There is insufficient system area memory.

Function #28: Close socket

Syntax: `CALL "SOCKET.FN3" 28 SOCKFD%`

Description: This function closes the specified socket identifier.

BSD4.4 socket API equivalent: This function is equivalent to the BSD4.4 socket API close() function.

Parameters: `SOCKFD%` Socket identifier

Return value: (None)

Run-time errors:

Error code	Meaning
105h	Power-off detected. (BHT-7500S only)
209h	Socket identifier is invalid.
225h	The last close operation for the specified socket is not complete.
23Ah	The specified TCP socket has been closed.
23Ch	The connection attempt has timed out.

Function #40: Specify TCP/IP communications pathway

Syntax: `CALL "SOCKET.FN3" 40 IFTYPE%, LAYERMODE%, INTERFACE%`

Description: This function specifies the TCP/IP communications pathway from the specified communications device and link layer.

Parameters:

`IFTYPE%`	Communications device
`LAYERMODE%`	Link layer

The communications device (`IFTYPE%`) must be 2, the value indicating a COM3 (wireless) communications device.

The link layer (`LAYERMODE%`) must be 2, the value indicating an Ethernet client.

Return value: `INTERFACE%` Communications pathway

Run-time errors:

Error code	Meaning
100h	Cannot specify communications pathway.

Function #41: Connect TCP/IP communications pathway with system settings

Syntax: `CALL "SOCKET.FN3" 41 INTERFACE%`

Description: This function connects the TCP/IP communications pathway based on the system settings.

Parameters:

`INTERFACE%`	Communications pathway

Return value: (None)

Run-time errors:

Error code	Meaning
`34h`	Communications device file not open.
`101h`	Cannot connect to communications pathway.
`102h`	Communications pathway not specified.
`103h`	Communications pathway already connected.
`105h`	Power-off detected. (BHT-7500S only)
`216h`	A parameter is invalid.

Function #42: Connect TCP/IP communications pathway with user settings

Syntax: `CALL "SOCKET.FN3" 42 INTERFACE%, IPADDRESS$, SUBNET-MASK$, GATEWAY$`

Description: This function connects the TCP/IP communications pathway based on the supplied user settings.

Parameters:

`INTERFACE%`	Communications pathway
`IPADDRESS$`	Internet address in dotted quad notation
`SUBNETMASK$`	Subnet mask in dotted quad notation
`GATEWAY$`	Default gateway in dotted quad notation

Return value: (None)

Run-time errors:

Error code	Meaning
`34h`	Communications device file not open.
`101h`	Cannot connect to communications pathway.
`102h`	Communications pathway not specified.
`103h`	Communications pathway already connected.
`105h`	Power-off detected. (BHT-7500S only)
`216h`	A parameter is invalid.

Function #43: Disconnect TCP/IP communications pathway

Syntax: `CALL "SOCKET.FN3" 43 INTERFACE%`

Description: This function disconnects the specified TCP/IP communications pathway.

Parameters: `INTERFACE%` Communications pathway

Return value: (None)

Run-time errors:

Error code	Meaning
`104h`	Communications pathway already disconnected.
`105h`	Power-off detected. (BHT-7500S only)
`216h`	A parameter is invalid.

Function #44: Get TCP/IP system settings

Syntax: `CALL "SOCKET.FN3" 44 PARA%, DATA$`

Description: This function gets the current setting for the specified TCP/IP system settings.

Parameters: PARA% Setting number

Return value: `DATA$` Current setting for TCP/IP system settings

Correspondence tables:

Setting Number (PARA%)	Description	Values for Setting (`DATA$`)
1	IP address	Character string in dotted quad notation, maximum 15 bytes
2	Subnet mask	Character string in dotted quad notation, maximum 15 bytes
3	Default gateway	Character string in dotted quad notation, maximum 15 bytes

Function #45: Set TCP/IP system settings

Syntax: `CALL "SOCKET.FN3" 45 PARA%, DATA$`

Description: This function sets the specified TCP/IP system settings to the new value.

Parameters:

`PARA%`	Setting number
`DATA$`	New setting for TCP/IP system settings

Return value: (None)

Correspondence tables:

See Table under function #44.

Function #46: Get TCP socket status

Syntax: `CALL "SOCKET.FN3" 46 SOCKFD%, PATTERN%, TIMEOUT%, RESULT%`

Description: This function waits until the specified TCP socket is in the specified state or the specified time elapsed.

Parameters:

`SOCKFD%`	Socket identifier
`PATTERN%`	Desired socket state
`TIMEOUT%`	Waiting period (in milliseconds, 100 ms resolution)

The socket state (`PATTERN%`) must be &h0020, the value indicating that the opposite end has sent FIN to close the socket. Only TCP sockets support this function.

Note: Specifying an invalid state sometimes stops processing.

`TIMEOUT%` must be one of the following values:

-1	No timeout
0	Read current state
1 to 32767	Wait specified time (timer resolution: 100 ms)

Return value: `RESULT%` Current socket state

`RESULT%` contains the current socket state. After a timeout, `RESULT%` contains 0.

Run-time errors:

Error code	Meaning
`105h`	Power-off detected. (BHT-7500S only)
`209h`	Socket identifier is invalid.
`216h`	A parameter is invalid.

18.6 FTP Library (`FTP.FN3`)

18.6.1 Overview

■ String Variables

The following are the string variables used by this library together with their memory requirements.

Description	Name	Size in Bytes
Version information	`VERSION$`	7
Server IP address	`SERV.IP`	15
Login user name	`USERNAME$`	0 to 16
Login password	`PASSWORD$`	0 to 16
Directory names	`CURDIR$ NEWDIR$`	0 to 255

File names	`SERV.FNAME$`
	`CLNT.FNAME$`
	`OLD.FNAME$`
	`NEW.FNAME$`
		0 to 12
		0 to 12
		0 to 12
		0 to 12

Field lengths	`FLD$`	1 to 64 (48)
FTP parameter	`FTP.PARA$`	

Function Number	Description	FTP Commands
`0`	Get FN3 version information	---
`1`	Open FTP client session with system settings	USER/PASS
`2`	Open FTP client session with user settings	USER/PASS
`3`	Close FTP client session	---
`4`	Get current directory on FTP server	PWD
`5`	Change current directory on FTP server	CWD
`6`	Download file from FTP server	RETR
`7`	Upload file to FTP server	STOR/APPE
`8`	Get FTP system settings	---
`9`	Set FTP system settings	---
`10`	Change file name on FTP server	RNFR/RNTO
`11`	Set port number for file transfer	PORT
`12`	Delete file from FTP server	DELE

See also the run-time errors for the `FTP.FN3` library.

■ Reply Codes

The messages that FTP servers send during and after FTP operations vary, but servers all use the same reply codes. (See Table.) All function numbers therefore supply these as their return value (REPLY%).

Reply Codes	Description
110	Restart marker replay.
120	Service ready in nnn minutes.
125	Data connection already open; transfer starting.
150	File status okay; about to open data connection.
200	Command okay.
202	Command not implemented, superfluous at this site.
211	System status, or system help reply.
212	Directory status.
213	File status.
214	Help message. On how to use the server or the meaning of a particular non-standard command. This reply is useful only to the human user.
215	NAME system type. Where NAME is an official system name from the list in the Assigned Numbers document.
220	Service ready for new users.
221	Service closing control connection. Logged out if appropriate.
225	Data connection open; no transfer in progress.
226	Closing data connection. Requested file action successful (for example, file transfer or file abort).
227	Entering Passive Mode (h1, h2, h3, h4, p1, p2).
230	User logged in, proceed.
250	Requested file action okay, completed.
257	“PATHNAME” created.
331	User name okay, need password.
332	Need account for login.
350	Requested file action pending further information.
421	Service not available, closing control connection. This may be a reply to any command if the service knows it must shut down.

Reply Codes	Description
425	Can’t open data connection.
426	Connection closed; transfer aborted.
450	Requested file action not taken. File unavailable (e.g., file busy).
451	Requested action aborted: local error in processing.
452	Requested action not taken. Insufficient storage space in system.
500	Syntax error, command unrecognized. This may include errors such as command line too long.
501	Syntax error in parameters or arguments.
502	Command not implemented.
503	Bad sequence of commands.
504	Command not implemented for that parameter.
530	Not logged in.
532	Need account for storing files.
550	Requested action not taken. File unavailable (e.g., file not found, no access).
551	Requested action aborted: page type unknown.
552	Requested file action aborted. Exceeded storage allocation (for current directory or dataset).
553	Requested action not taken. File name not allowed.

18.6.2 Detailed Function Specifications

Function #0: Get FTP.FN3 version information

Syntax: `CALL "FTP.FN3" 0 VERSION$`

Description: This function returns the FTP.FN3 library version in `VERSION$`.

Parameters: (None)

Return value: `VERSION$` Version information, 7 characters, fixed length

Function #1: Open FTP client session with system settings

Syntax: `CALL "FTP.FN3" 1 FTPHANDLE%, REPLY%`

Description: This function opens an FTP client session using the system settings.

Parameters: (None)

Return value:

`FTPHANDLE%`	FTP client handle, for use by following functions
`REPLY%`	Server response to FTP command

Run-time errors:

Error code	Meaning
`105h`	Power-off detected. (BHT-7500S only)
`110h`	Response other than 2XX received.
`20Dh`	Attempt to connect to different FTP server without disconnecting.
`216h`	The FTP client handle is invalid.
`239h`	The specified socket is not connected.
`23Ch`	The connection attempt has timed out.

Function #2: Open FTP client session with user settings

Syntax: `CALL "FTP.FN3" 2 FTPHANDLE%, SERV.IP$, USERNAME$, PASSWORD$, REPLY%`

Description: This function opens an FTP client session based on the supplied user settings.

Parameters:

`SERV.IP$`	FTP server IP address in dotted quad notation
`USERNAME$`	User name for FTP authentication
`PASSWORD$`	Password for FTP authentication

Return value:

`FTPHANDLE%`	FTP client handle, for use by following functions
`REPLY%`	Server response to FTP command

Run-time errors:

Error code	Meaning
`105h`	Power-off detected. (BHT-7500S only)
`110h`	Response other than 2XX received.
`20Dh`	Attempt to connect to different FTP server without disconnecting.
`216h`	The FTP client handle is invalid.
`239h`	The specified socket is not connected.
`23Ch`	The connection attempt has timed out.

Function #3: Close FTP client session

Syntax: `CALL "FTP.FN3" 3 FTPHANDLE%, REPLY%`

Description: This function closes the specified FTP client session.

Parameters: `FTPHANDLE%` FTP client handle

Return value: `REPLY%` Server response to FTP command

Run-time errors:

Error code	Meaning
`105h`	Power-off detected. (BHT-7500S only)
`110h`	Response other than 2XX received.
`216h`	The FTP client handle is invalid.
`239h`	The specified socket is not connected.

Function #4: Get current directory on FTP server

Syntax: `CALL "FTP.FN3" 4 FTPHANDLE%, CURDIR$, REPLY%`

Description: This function gets the current directory on the FTP server.

Parameters: `FTPHANDLE%` FTP client handle

Return value: `CURDIR$` FTP server current directory
`REPLY%` Server response to FTP command

Run-time errors:

Error code	Meaning
`105h`	Power-off detected. (BHT-7500S only)
`110h`	Response other than 2XX received.
`216h`	The FTP client handle is invalid.
`239h`	The specified socket is not connected.
`295h`	There is no user for login request.

Note: The directory specification (`CURDIR$`) is limited to 255 bytes, so do not use longer directory names on the server.

Function #5: Change current directory on FTP server

Syntax: `CALL "FTP.FN3" 5 FTPHANDLE%, NEWDIR$, REPLY%`

Description: This function changes the current directory on the FTP server.

Parameters:

`FTPHANDLE%`	FTP client handle
`NEWDIR$`	New directory

Return value:

`REPLY%`	Server response to FTP command

Run-time errors:

Error code	Meaning
`105h`	Power-off detected. (BHT-7500S only)
`110h`	Response other than 2XX received.
`216h`	The FTP client handle is invalid.
`239h`	The specified socket is not connected.
`295h`	There is no user for login request.

Function #6: Download file from FTP server

Syntax: `CALL "FTP.FN3" 6 FTPHANDLE%, SERV.FNAME$, CLNT.FNAME$, CRLF.TYPE%, CRLF.MODE%, REPLY% [,FLD$] [,DISP.MODE%]`

Description: This function downloads, from the current directory on the FTP server to the handy terminal, the specified file using the specified parameters.

Parameters:

`FTPHANDLE%`	FTP client handle
`SERV.FNAME$`	Name of file to download from FTP server
`CLNT.FNAME$`	Name for file on handy terminal. Leaving this unspecified ("") uses the name in `SERV.FNAME$` instead.

Note: `SERV.FNAME$` and `CLNT.FNAME$` must have the same type (file extension): user program (.PD3), extension library (.FN3 or .EX3), or data file (all other extensions). Otherwise, the run-time error 32h is the result.

`CRLF.TYPE%`	Line delimiter

0	CR-LF combination (Treat CR-LF combinations as delimiters. Use this value when the data file delimits records with CR-LF combinations.)
1	LF (Treat LFs as delimiters. Use this value when the data file delimits its records with LFs.)
2	CR (Treat CRs as delimiters. Use this value when the data file delimits its records with CRs.)
3	None Use this value when the data file does not delimit records.

`CRLF.MODE%`	Treatment of line delimiters inside records and trailing spaces in fields Note: `CRLF.MODE%` will be ignored for files except data files.

0	Treat line delimiters inside records as SEPARATORS. TRIM trailing spaces in fields.
1	Treat line delimiters inside records as DATA. TRIM trailing spaces in fields.
10	Treat line delimiters inside records as SEPARATORS. RETAIN trailing spaces in fields.
11	Treat line delimiters inside records as DATA. RETAIN trailing spaces in fields.

`FLD$`	Field lengths in bytes. Delimit the field length specifications with commas (,) or semicolons (;). (This parameter applies only to downloaded data files.) "<field length 1> [,<field length 2>,... <field length n>]" (n=1 to 16, field length = 1 to 254)
`DISP.MODE%`	Flag controlling a progress display consisting of an 8-digit number giving the number of bytes transferred

0	Disable
1	Enable

Return value: `REPLY%` Server response to FTP command

Example: Downloading a data file

```
SERV.FNAME$ = "MASTER.DAT"    ' File name on server
CLNT.FNAME$ = ""              ' Name for file on the handy terminal
                              ' Same as on server
CRLF.TYPE% = 1                ' Server line delimiter: LF
CRLF.MODE% = 0                ' Data composition
                              ' There are no line delimiters in the data.
FLD$ = "3, 2, 1"              ' Field lengths: 3, 2, 1
CALL "FTP.FN3" 6 FTPHANDLE%, SERV.FNAME$, CLNT.FNAME$, CRLF.TYPE%, _
CRLF.MODE%, REPLY%, FLD$
```

Example: Downloading a program file, with progress display

```
SERV.FNAME$ = "SAMPLE.PD3"    ' File name on server
CLNT.FNAME$ = ""              ' Name for file on the handy terminal
                              ' Same as on server
CRLF.TYPE% = 0                ' Server line delimiter: CR-LF combination
CRLF.MODE% = 0                ' Data composition: Will be ignored for
                              ' files except data files
DISP.MODE% = 1                ' Enable progress display
CALL "FTP.FN3" 6 FTPHANDLE%, SERV.FNAME$, CLNT.FNAME$, CRLF.TYPE%, _
CRLF.MODE%, REPLY%, DISP.MODE%
```

Run-time errors:

Error code	Meaning
`02h`	Syntax error (Incorrect file name).
`05h`	Number of field items or number of digits in a field out of the range.
`07h`	Insufficient memory space.
`32h`	Wrong file type.
`33h`	Invalid text received.
`37h`	File already open.
`39h`	Too many files.
`3Ch`	Record exceeds 255 bytes.
`3Dh`	Field mismatch error.
`41h`	File damaged.
`47h`	User break with cancel (C) key.
`49h`	Invalid program file received. (Invalid program size. Do not download user programs that have been run through Kanji conversion utilities.)
`105h`	Power-off detected. (BHT-7500S only)
`110h`	Response other than 2XX received.
`111h`	File not closed.
`216h`	The FTP client handle is invalid.
`239h`	The specified socket is not connected.
`295h`	There is no user for login request.

Function #7: Upload file to FTP server

Syntax: `CALL "FTP.FN3" 7 FTPHANDLE%, SERV.FNAME$, CLNT.FNAME$, CRLF.TYPE%, UP.MODE%, REPLY% [,DISP.MODE%]`

Description: This function uploads, from the handy terminal to the current directory on the FTP server, the specified file using the specified parameters.

Parameters:

`FTPHANDLE%`	FTP client handle
`SERV.FNAME$`	Name for file on FTP server. Leaving this unspecified ("") uses the name in `CLNT.FNAME$` instead.
`CLNT.FNAME$`	Name of file to upload to FTP server.
`CRLF.TYPE%`	Line delimiter (See description under function #6 above.)
`UP.MODE%`	Flag controlling treatment of existing files

0	Overwrite existing file
1	Append to existing file. Create new file if necessary.

`DISP.MODE%`	Flag controlling a progress display consisting of an 8-digit number giving the number of bytes transferred

0	Disable
1	Enable

Return value: `REPLY%` Server response to FTP command

Example: Uploading data file

```
RCLNT.FNAME$ = "MASTER1.DAT"   ' Name of file on handy terminal
SERV.FNAME$ = ""               ' Name on server
                               ' Same as on handy terminal
CRLF.TYPE% = 0                 ' Server line delimiter: CR-LF combination
UP.MODE% = 1                   ' Upload mode: Append
CALL "FTP.FN3" 7 FTPHANDLE%, SERV.FNAME$, CLNT.FNAME$, CRLF.TYPE%, _
UP.MODE%, REPLY%
```

Example: Uploading program file, with progress display

```
CLNT.FNAME$ = "SAMPLE.PD3"     ' Name of file on handy terminal
SERV.FNAME$ = ""               ' Name on server
                               ' Same as on handy terminal
CRLF.TYPE% = 0                 ' Server line delimiter: CR-LF combination
UP.MODE% = 0                   ' Upload mode: Overwrite
DISP.MODE% = 1                 ' Enable progress display
CALL "FTP.FN3" 7 FTPHANDLE%, SERV.FNAME$, CLNT.FNAME$, CRLF.TYPE%, _
UP.MODE%, REPLY%, DISP.MODE%
```

Run-time errors:

Error code	Meaning
`35h`	File not found.
`37h`	File already open.
`47h`	User break with cancel (C) key.
`105h`	Power-off detected. (BHT-7500S only)
`110h`	Response other than 2XX received.
`111h`	File not closed.
`216h`	The FTP client handle is invalid.
`239h`	The specified socket is not connected.
`295h`	There is no user for login request.

Function #8: Get FTP system settings

Syntax: `CALL "FTP.FN3" 8 PARA%,` *`ftp.para`*
where `ftp.para` is `FTP.PARA%` or `FTP.PARA$`

Description: This function gets the current setting for the specified FTP system settings.

Parameters: PARA% Setting number

Return value: *`ftp.para`* Current setting for FTP system settings of type integer/ string (`FTP.PARA%/FTP.PARA$`)

Correspondence tables:

Setting Number (PARA%)	Description	Values for Setting (`FTP.PARA%`)
5	Line delimiter	0 (CR-LF), 1 (LF), 2 (CR), 3 (None)
6	Treatment of line delimiters inside records	0 (separators), 1 (data)
7	Upload mode	0 (overwrite), 1 (append)
8	Progress display	0 (disabled), 1 (enabled)

Setting Number (PARA%)	Description	Values for Setting (`FTP.PARA$`)
1	IP address for FTP server	Character string in dotted quad notation, maximum 15 bytes
2	User name for FTP authentication	Character string, maximum 16 bytes

Setting Number (PARA%)	Description	Values for Setting (FTP.PARA$)
3	Password for FTP authentication	Character string, maximum 16 bytes
4	Initial directory on FTP server	character string, a maximum of 63 bytes long

Function #9: Set FTP system settings

Syntax: `CALL "FTP.FN3" 9 PARA%, ftp.para`
where `ftp.para` is `FTP.PARA%` or `FTP.PARA$`

Description: This function sets the specified FTP system settings to the new value.

Parameters:

`PARA%`	Setting number
`ftp.para`	New setting for FTP system settings of type integer/string (`FTP.PARA%/FTP.PARA$`)

Return value: (None)

Correspondence tables:

See Table under function #8.

Function #10: Change file name on FTP server

Syntax: `CALL "FTP.FN3" 10 FTPHANDLE%, OLD.FNAME$, NEW.FNAME$, REPLY%`

Description: This function changes the name of a file in the current directory on the FTP server.

Parameters:

`FTPHANDLE%`	FTP client handle
`OLD.FNAME$`	Name before change
`NEW.FNAME$`	Name after change

Return value: `REPLY%` Server response to FTP command

Run-time errors:

Error code	Meaning
105h	Power-off detected. (BHT-7500S only)
110h	Response other than 2XX received.
216h	The FTP client handle is invalid.
239h	The specified socket is not connected.
295h	There is no user for login request.

Function #11: Set port number for file transfer

Syntax: `CALL "FTP.FN3" 11` *`FTPHANDLE%, PORT%`*

Description: This function sets a port number specified by *`PORT%`* for file transfer.

Parameters:

FTPHANDLE%	FTP client handle
PORT%	Port number

Return value: (None)

Run-time errors:

Error code	Meaning
105h	Power-off detected. (BHT-7500S only)
216h	The FTP client handle is invalid.
239h	The specified socket is not connected.
295h	There is no user for login request.

Function #12: Delete file from FTP server

Syntax: `CALL "FTP.FN3" 12` *`FTPHANDLE%, SERV.FNAME$, REPLY%`*

Description: This function deletes a file specified by *`SERV.FNAME$`* from the FTP server.

Parameters:

FTPHANDLE%	FTP client handle
SERV.FNAME$	File name to be deleted

Return value: *REPLY%* Server response to FTP command

Run-time errors:

Error code	Meaning
105h	Power-off detected. (BHT-7500S only)
110h	Response other than 2XX received.
216h	The FTP client handle is invalid.
239h	The specified socket is not connected.
295h	There is no user for login request.

Appendices

CONTENTS

Appendix A Error Codes and Error Messages

A1. Run-time Errors

Error code	Meaning
00h	Internal system error
01h	NEXT without FOR
02h	Syntax error
03h	RETURN without GOSUB
04h	Out of DATA (No DATA values remain to be read by the READ statement.)
05h	Parameter out of the range
06h	The operation result is out of the allowable range.
07h	Insufficient memory space (Too deep nesting, etc.)
08h	Array not defined
09h	Subscript out of range (An array subscript is out of the array. Or the array is referenced by different dimensions.)
0Ah	Duplicate definition (An array is double defined.)
0Bh	Division by zero
0Ch	CASE and END SELECT without SELECT
0Dh	END DEF or EXIT DEF statement executed outside the DEF FN statement block
0Fh	String length out of the range
10h	Expression too long or complex
14h	RESUME without error (RESUME statement occurs before the start of an error-handling routine.)
1Fh	Function number out of the range (in CALL statement)
32h	File type mismatch
33h	Received text format not correct
34h	Bad file name or number (A statement uses the file number of an unopened file.)
35h	File not found

Error code	Meaning
36h	Improper file type (The statement attempts an operation that conflicts with the file type--data file, communications device file, or bar code device file.)
37h	File already open (An OPEN statement executed for the already opened file.)
38h	The file name is different from that in the receive header.
39h	Too many files
3Ah	File number out of the range
3Bh	The number of the records is greater than the defined maximum value.
3Ch	FIELD overflow (A FIELD statement specifies the record length exceeding 255 bytes.)
3Dh	A FIELD statement specifies the field width which does not match one that specified in file creation.
3Eh	FIELD statement not executed yet (A PUT or GET statement executed without a FIELD statement.)
3Fh	Bad record number (The record number is out of the range.)
40h	Parameter not set (ID not set)
41h	File damaged
42h	File write error (You attempted to write onto a read-only file.)
43h	Not allowed to access data in the flash ROM
44h	No empty area of the specified size in the RAM
45h	Device files prohibited from opening concurrently
46h	Communications error
47h	Abnormal end of communications or termination of communications by the Clear key
48h	Device timeout (No CS signal has been responded within the specified time period.)
49h	Received program file not correct
F0h	Mismatch parameter number
F1h	Mismatch parameter type
F2h	Insufficient string variable storage area
100h	Cannot specify communications pathway.
101h	Cannot connect to communications pathway.
102h	Communications pathway not specified.

Error code	Meaning
103h	Communications pathway already connected.
104h	Communications pathway already disconnected.
105h	Power-off detected.
110h	Response other than 2XX received.
111h	File not closed.
201h	Cannot connect to socket.
209h	Socket identifier is invalid.
20Dh	Attempt to connect to different FTP server without disconnecting.
216h	A parameter is invalid. The FTP client handle is invalid. A parameter is invalid, or the socket is already bound.
218h	Too many sockets.
224h	The socket is being assigned an address.
225h	The last close operation for the specified socket is not complete.
228h	The maximum number of bytes to receive is too small.
229h	The specified socket does not match the connection target socket.
22Ah	This option is not recognized at the specification level.
22Bh	This protocol family does not support the specified protocol type and protocol.
22Fh	The specified address family is invalid for this socket.
230h	The specified address is already in use.
231h	The specified address is invalid.
236h	An RST from the opposite end has forced connection.
237h	There is insufficient system area memory.
238h	The specified socket is already connected.
239h	The specified socket is not connected.
23Ah	The specified TCP socket has been closed.
23Ch	The connection attempt has timed out.
23Dh	Failed to connect.
241h	There is no connection pathway to the host for TCP socket.
295h	There is no user for login request.
400h	Failed to get the setting value. (Failed to set the value.)
401h	Failed to open a wireless communications device file.

A2. Compilation Errors

■ Fatal Errors

Error code & Message	
fatal error 1:	Out of memory
fatal error 2:	Work file I/O error
fatal error 3:	Object file I/O error
fatal error 4:	Token file I/O error
fatal error 5:	Relocation information file I/O error
fatal error 6:	Cross reference file I/O error
fatal error 7:	Symbol file I/O error
fatal error 8:	Compile list file I/O error
fatal error 9:	Debug information file I/O error (source-address)
fatal error 10:	Debug information file I/O error (label-address)
fatal error 11:	Debug information file I/O error (variable-intermediate code)
fatal error 12:	Out of disk space for work file
fatal error 13:	Out of disk space for object file
fatal error 14:	Out of disk space for token file
fatal error 15:	Out of disk space for relocation information file
fatal error 16:	Out of disk space for cross reference file
fatal error 17:	Out of disk space for symbol file
fatal error 18:	Out of disk space for compile list file
fatal error 19:	Out of disk space for debug information file (source-address)
fatal error 20:	Out of disk space for debug information file (label-address)
fatal error 21:	Out of disk space for debug information file (variable-intermediate code)
fatal error 22:	Source file I/O error
fatal error 23:	Cannot find XXXX.SRC
fatal error 24:	Error count exceeds 500
fatal error 25:	Out of memory (internal labels exceed 3000)
fatal error 26:	Control structure nesting exceeds 30
fatal error 27:	Expression type stack exceeds 50
fatal error 28:	Program too large (Object area overflow)

Error code & Message	
`fatal error 29:`	`Out of memory for cross reference`
`fatal error 30:`	`Cannot find include file`
`fatal error 31:`	`Cannot nest include file`
`fatal error 32:`	`Internal memory allocation error (tag list buffer) [function name]`
`fatal error 33:`	`(Preprocess) Source file I/O error`
`fatal error 34:`	`(Preprocess) Internal memory overflow`
`fatal error 35:`	`(Preprocess) Macro work file I/O error`
`fatal error 36:`	`(Preprocess) Macro double defined [Macro name]`
`fatal error 37:`	`(Preprocess) Internal memory overflow (unread buffer)`
`fatal error 38:`	`(Preprocess) Memory allocation error`
`fatal error 39:`	`(Preprocess) Macro circular reference [Macro name]`

■ Syntax Errors

Error code & Message	
error 1:	Improper label format
error 2:	Improper label name (redefinition, variable name, or reserved word used)
error 3:	’"’missing
error 4:	Improper expression
error 5:	Variable name redefinition (common variable already defined as label name or variable name)
error 6:	Variable name redefinition (register variable already defined as label name or variable name)
error 7:	Variable name redefinition (variable already defined as label name, non-array string work variable, register variable, or common variable)
error 8:	Too many variables (work integer non-array)
error 9:	Too many variables (work float non-array)
error 10:	Too many variables (work string non-array)
error 11:	Too many variables (register integer non-array)
error 12:	Too many variables (register float non-array)
error 13:	Too many variables (register string non-array)
error 14:	Too many variables (common integer non-array)
error 15:	Too many variables (common float non-array)
error 16:	Too many variables (common string non-array)
error 17:	Too many variables (work integer array)
error 18:	Too many variables (work float array)
error 19:	Too many variables (work string array)
error 20:	Too many variables (register integer array)

Error code & Message	
error 21:	Too many variables (register float array)
error 22:	Too many variables (register string array)
error 23:	Too many variables (common integer array)
error 24:	Too many variables (common float array)
error 25:	Too many variables (common string array)
error 26:	Too many variables (work integer array, two-dimensional)
error 27:	Too many variables (work float array, two-dimensional)
error 28:	Too many variables (work string array, two-dimensional)
error 29:	Too many variables (register integer array, two-dimensional)
error 30:	Too many variables (register float array, two-dimensional)
error 31:	Too many variables (register string array, two-dimensional)
error 32:	Too many variables (common integer array, two-dimensional)
error 33:	Too many variables (common float array, two-dimensional)
error 34:	Too many variables (common string array, two-dimensional)
error 35:	Source line too long
error 36:	
error 37:	
error 38:	
error 39:	
error 40:	
error 41:	Value out of range for integer constant
error 42:	Value out of range for float constant
error 43:	Value out of range for integer constant (hexadecimal expression)
error 44:	Improper hexadecimal expression
error 45:	Symbol too long

Error code & Message	
error 46:	
error 47:	
error 48:	
error 49:	
error 50:	Incorrect use of IF...THEN...ELSE...ENDIF
error 51:	Incomplete control structure (IF...THEN...ELSE...ENDIF)
error 52:	Incorrect use of FOR...NEXT
error 53:	Incomplete control structure (FOR...NEXT)
error 54:	Incorrect FOR index variable
error 55:	Incorrect use of SELECT...CASE...END SELECT
error 56:	Incomplete control structure (SELECT...CASE...END SELECT)
error 57:	Incorrect use of WHILE...WEND
error 58:	Incomplete control structure (WHILE...WEND)
error 59:	Incorrect use of DEF FN...EXIT DEF...END DEF
error 60:	Incomplete control structure (DEF FN...END DEF)
error 61:	Cannot use DEF FN in control structure
error 62:	Operator stack overflow
error 63:	Inside function definition
error 64:	Function redefinition
error 65:	Function definitions exceed 200
error 66:	Arguments exceed 50
error 67:	Total arguments exceed 500
error 68:	Mismatch argument type or number
error 69:	Function undefined
error 70:	Label redefinition
error 71:	Syntax error
error 72:	Variable name redefinition
error 73:	Improper string length
error 74:	Improper array elements number
error 75:	Out of space for register variable area
error 76:	Out of space for work, common variable area

Error code & Message	
error 77:	Initial string too long
error 78:	Array symbols exceed 30 for one DIM, GLOBAL, or PRIVATE statement
error 79:	Record number out of range (1 to 32767)
error 80:	Label undefined
error 81:	Must be DATA statement label (in RESTORE statement)
error 82:	’(’ missing
error 83:	’)’ missing
error 84:	’]’ missing
error 85:	’,’ missing
error 86:	’;’ missing
error 87:	’DEF’ missing
error 88:	’TO’ missing
error 89:	’INPUT’ missing
error 90:	’{’ missing
error 91:	Improper initial value for integer variable (not integer or out of range)
error 92:	Incorrect use of SUB, EXIT SUB, or END SUB
error 93:	Incomplete control structure (SUB...END SUB)
error 94:	Cannot use SUB statement in control structure
error 95:	Incorrect use of FUNCTION, EXIT FUNCTION, or END FUNCTION
error 96:	Incomplete control structure (FUNCTION...END FUNCTION)
error 97:	Cannot use FUNCTION statement in control structure
error 98:	Incorrect use of CONST

■ Linking Errors

Error Message
PRC area size different
Out of space in REG area
Out of space in PRD area
Cannot open project file
Cannot open object file [object name]
Cannot open MAP file
Cannot open PD3 file [PD3 filename]
Cannot close PD3 file [PD3 filename]
Write error to PD3 file [PD3 filename]
Seek error: Cannot move to the filename position
Seek error: Cannot move to the head of the block
Filename area too large
Symbolname area too large
Too many records in symbol table
Too many modules
Too many libraries
Too many objects
Failed to allocate memory in TAG area
Failed to allocate memory in link TAG area
Undefined value set to variable type [Value at variable type]
Undefined value set to tag type [Value at tag type]
Module [modulename] not defined
Symbol [symbolname] not defined
Cannot register symbol
More than one symbol type [variable type*] existing
Defined [variable types*] over the maximum limit
More than one symbol [symbolname] defined
Number of descriptors over the limit
Common variable [variablename] defined out of main module
Common data area overflow
Work data area overflow
Symbol name area overflow

Error Message
`Non-array integer register variable area overflow`
`Non-array float register variable area overflow`
`Register memory pool area overflow`
`Failed to set up initial setting of register data`

* To the [Variable type], any of the following character strings applies:

- Non-array integer common variable
- Non-array float common variable
- Non-array string common variable
- Non-array integer work variable
- Non-array float work variable
- Non-array string work variable
- Non-array integer register variable
- Non-array float register variable
- Non-array string register variable
- One-dimensional array integer common variable
- One-dimensional array float common variable
- One-dimensional array string common variable
- One-dimensional array integer work variable
- One-dimensional array float work variable
- One-dimensional array string work variable
- One-dimensional array integer register variable
- One-dimensional array float register variable
- One-dimensional array string register variable
- Two-dimensional array integer common variable
- Two-dimensional array float common variable
- Two-dimensional array string common variable
- Two-dimensional array integer work variable
- Two-dimensional array float work variable
- Two-dimensional array string work variable
- Two-dimensional array integer register variable
- Two-dimensional array float register variable
- Two-dimensional array string register variable

■ Library Errors

Error Message
`Cannot find object to be deleted [objectname]`
`Designated object already existing [objectname]`
`Cannot find object to be updated [objectname]`
`Module already defined [modulename]`
`Filename area too large`
`Too many block information pieces`
`Cannot open library file`
`Seek error: Cannot move to the filename position`
`Seek error: Cannot move to the head of the block`

NOTE No error code precedes any linking error or library error.

Appendix B
Reserved Words

The following list shows reserved words (keywords) of BHT-BASIC. Any of these words must not be used as a variable name or label name.

A	ABS	F	FIELD	P	POS
	AND		FN		POWER
	APLOAD		FOR		PRINT
	AS		FRE		PRINT#
	ASC	G	GET		PUT
B	BCC$		GO	R	READ
	BEEP		GOSUB		RECORD
C	CALL		GOTO		REM
	CASE	H	HEX		RESTORE
	CHAIN	I	IF		RESUME
	CHKDGT		$INCLUDE		RETURN
	CHR		INKEY		RIGHT$
	CLFILE		INP	S	SCREEN
	CLOSE		INPUT		SEARCH
	CLS		INSTR		SELECT
	CODE		INT		SEP
	COMMON	K	KEY		SOH
	CONT		KILL		STEP
	COUNTRY		KPLOAD		STR
	CSRLIN	L	LEFT		STX
	CURSOR		LEN	T	THEN
D	DATA		LET		TIME
	DATE$		LINE		TIMEA
	DEF		LOC		TIMEB
	DEFREG		LOCATE		TIMEC
	DIM		LOF		TO
E	ELSE	M	MARK	U	USING
	END		MID	V	VAL
	EOF		MOD	W	WAIT
	ERASE	N	NEXT		WEND
	ERL		NOT		WHILE
	ERR	O	OFF	X	XFILE
	ERROR		ON		XOR
	ETB		OPEN		
	ETX		OR		
	EXIT		OUT		

Appendix C
Character Sets

C1. Character Set

The table below lists the character set which the BHT can display on the LCD screen. It is based on the ASCII codes.

Lower 4 bits \ Upper 4 bits	0000 0	0001 1	0010 2	0011 3	0100 4	0101 5	0110 6	0111 7	1000 8	1001 9	1010 A	1011 B	1100 C	1101 D	1110 E	1111 F
0000 0	▣	␣	␣	0	@	P	`	p			␣	ー	タ	ミ	α	p
0001 1	▣	␣	!	1	A	Q	a	q			。	ア	チ	ム	ä	q
0010 2	◀	␣	″	2	B	R	b	r			「	イ	ツ	メ	β	θ
0011 3	▶	␣	#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
0100 4	⬍	␣	$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
0101 5	█	␣	%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
0110 6	↑	␣	&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
0111 7	↓	␣	’	7	G	W	g	w			ァ	キ	ヌ	ラ	q	π
1000 8	BS	C	(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
1001 9	␣	␣	)	9	I	Y	i	y			ゥ	ケ	ノ	ル	[illegible]	ч
1010 A	␣	␣	*	:	J	Z	j	z			ェ	コ	ハ	レ	i	千
1011 B	␣	␣	+	;	K	[	k	{			ォ	サ	ヒ	ロ	*	万
1100 C	␣	␣	,	<	L	＼	l	|			ャ	シ	フ	ワ	¢	円
1101 D	CR	␣	—	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
1110 E	␣	␣	.	>	N	^	n	~			ョ	セ	ホ	゛	n̄	␣
1111 F	␣	␣	／	?	O	_	o	␣			ッ	ソ	マ	゜	ö	█

NOTE 1: You can assign user-defined fonts to codes from 80h to 9Fh with `APLOAD` statement. (Refer to `APLOAD` statement in Chapter 14.)

NOTE2: Characters assigned to codes 20h to 7Fh are default national characters when the English message version is selected on the menu screen* in System Mode.

* Menu screen for selecting the message version

BHT Series	Menu screen
BHT-3000	Set Resume menu
BHT-4000/BHT-5000/BHT-6000/ BHT-6500/BHT-7000/BHT-7500	SET DISPLAY menu

They can be switched to other national characters (see Appendix C2) by `COUNTRY$` function. (Refer to `COUNTRY$` function in Chapter 15.)

NOTE 3: `BS` is a backspace code.

NOTE 4: `CR` is a carriage return code.

NOTE 5: `C` is a cancel code.

NOTE 6: ␣ is a space code.

C2. National Character Sets

You may switch characters assigned to codes 20h to 7Fh of the character set table listed in Appendix C1 to one of the national character sets by using the `COUNTRY$` function.

The default national character set is America (code A) or Japan (code J) depending upon the English or Japanese message version selected on the menu screen* in System Mode, respectively.

* Menu screen for selecting the message version

BHT Series	Menu screen
BHT-3000	Set Resume menu
BHT-4000/BHT-5000/BHT-6000/ BHT-6500/BHT-7000/BHT-7500	SET DISPLAY menu

Listed below are national characters which are different from the defaults.

(Hex.)

Country	Country code**	23	24	40	5B	5C	5D	5E	60	7B	7C	7D	7E	7F
America (Default)	A	#	$	@	[	\	]	^	`	{	|	}	~	␣
Denmark	D				Æ	Ø	Å						~	␣
England	E	£	$			\							~	␣
France	F			à			§						¨	␣
Germany	G			§	Ä	Ö	Ü						ß	␣
Italy	I					\				à				␣
Japan (Default)	J	#	$	@	[	¥	]	^	`	{	|	}	→	←
Norway	N		¤	É	Æ	Ø	Å	Ü						␣
Spain	S	Pt				Ñ	¿			¨		}	~	␣
Sweden	W		¤	É	Ä	Ö	Å	Ü						␣

** Refer to `COUNTRY$` function in Chapter 15.

```
COUNTRY$="countrycode"
```

NOTE 1: ␣ is a space code.

NOTE 2: Empty boxes in the above table are assigned the same characters as default ones listed in Appendix C1.

C3. Display Mode and Letter Size

■ Character frame and letter size

Screen mode	Font size		Character frame (W x H)	Letter size (W x H)
Single-byte ANK mode	Standard-size		6 x 8	5 x 7
	Small-size (BHT-6000/BHT-6500/ BHT-7000/BHT-7500)		6 x 6	5 x 5
Two-byte Kanji mode	Standard-size	Full-width Half-width	16 x 16 8 x 16	15 x 16 7 x 16
	Small-size (BHT-6000/BHT-6500/ BHT-7000/BHT-7500)	Full-width Half-width	12 x 12 6 x 12	11 x 12 5 x 12
Condensed two-byte Kanji mode (BHT-4000/BHT-5000)		Full-width Half-width	12 x 16 6 x 16	11 x 16 5 x 16

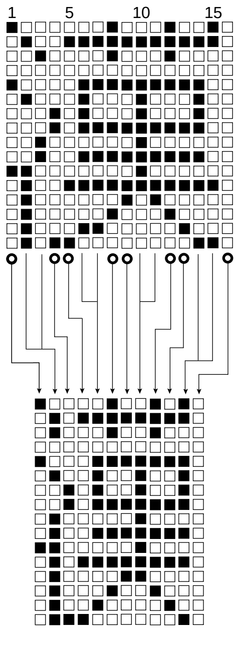
■ Generating the condensed two-byte Kanji patterns (BHT-4000/BHT-5000)

To display condensed two-byte Kanji characters, the Interpreter generates their font patterns by condensing the Kanji fonts stored in the Kanji ROM (in the BHT-4000) or by condensing the JIS Level 1 and Level 2 Kanji fonts stored in the flash ROM (in the BHT-5000).

The Interpreter can condense also Kanji patterns loaded by the KPLOAD statement. If the condensed two-byte Kanji mode is to be used, it is necessary to take into account the condensation when defining Kanji patterns.

The condensing process is as follows: The Interpreter ORs adjacent vertical two rows--2nd and 3rd rows, 6th and 7th rows, 10th and 11th rows, and 14th and 15th rows--to produce a single row each. Other rows will be displayed as they are.

In the figure shown at right, rows marked with ⚬ will be displayed as they are; adjacent two rows without ⚬ will be condensed into a single row.



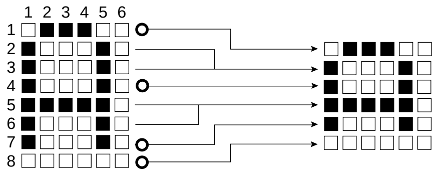
■ Generating the small-size font patterns

BHT-6000/BHT-6500

- Single-byte ANK characters

To display single-byte ANK characters in small size of fonts, their small-size font patterns stored in the flash ROM will be used and no condensation will take place.

For the patterns loaded by the APLOAD statement, the Interpreter condenses them as follows:



The Interpreter ORs adjacent horizontal two rows--2nd and 3rd rows and 5th and 6th rows--to produce a single row each. Other rows will be displayed as they are. In the figure shown above, rows marked with **o** will be displayed as they are; adjacent two rows without **o** will be condensed into a single row.

- Two-byte Kanji characters

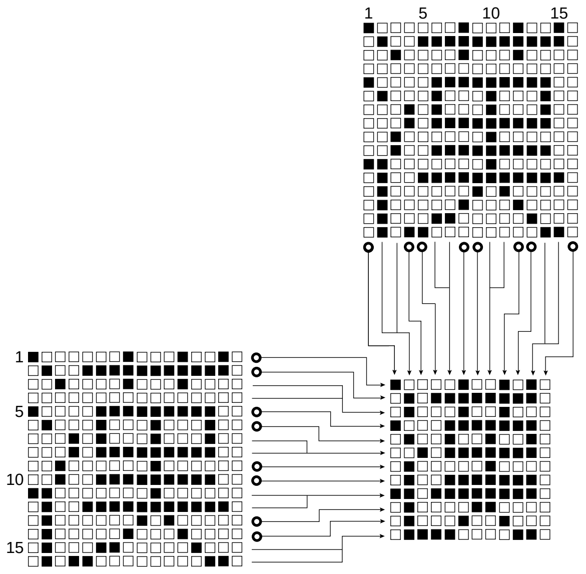
To display two-byte Kanji characters (full-width and half-width) in small size of fonts, the Interpreter generates their font patterns by condensing the JIS Level 1 and Level 2 Kanji fonts stored in the flash ROM. Also for Kanji patterns loaded by the KPLOAD statement, the Interpreter condenses them in the same way.

If Kanji patterns loaded by the KPLOAD statement are to be displayed in small size of fonts, it is necessary to take into account the condensation when defining Kanji patterns.

The condensing process is as follows:

The Interpreter ORs adjacent vertical two rows--2nd and 3rd rows, 6th and 7th rows, 10th and 11th rows, and 14th and 15th rows--to produce a single row each. Other rows will be displayed as they are. In the figure shown at right, rows marked with **o** will be displayed as they are; adjacent two rows without **o** will be condensed into a single row.

The Interpreter ORs adjacent horizontal two rows--3rd and 4th rows, 7th and 8th rows, 11th and 12th rows, and 15th and 16th rows--to produce a single row each. Other rows will be displayed as they are. In the figure shown below, rows marked with **o** will be displayed as they are; adjacent two rows without **o** will be condensed into a single row.

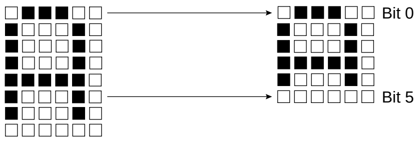


BHT-7000/BHT-7500

- Single-byte ANK characters

To display single-byte ANK characters in small size of fonts, their small-size font patterns stored in the flash ROM will be used and no condensation will take place.

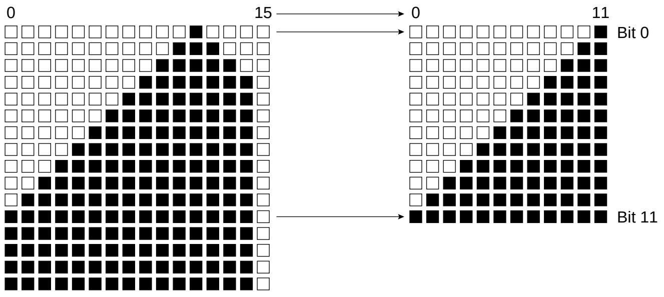
For the patterns loaded by the APLOAD statement, the Interpreter uses a total of 6 bits (bit 0 to 5) in each vertical row and ignores bits 6 and 7.



- Two-byte Kanji characters

To display two-byte Kanji characters (full-width and half-width) in small size of fonts, small-size font patterns of the JIS Level 1 and Level 2 Kanji stored in the user area of the memory will be used and no condensation will take place.

For the patterns loaded by the KPLOAD statement, the Interpreter uses a total of 12 bits (bit 0 to 11) each on the 1st to 11th elements and ignores the 12th to 15th elements and bits 12 to 15.



Appendix D
I/O Ports

D1. BHT-3000

■ Input Ports

A user program can monitor the hardware status through the input ports by using the `WAIT` statement or `INP` function. BHT-BASIC defines each of these ports as a byte. The table below lists the input ports and their monitoring function in the BHT.

Port No.	Bit assignment [*1]	Monitors the following:		
0	0	Keyboard buffer	0: No data	1: Data stored
	1	Barcode buffer	0: No data	1: Data stored
	2	Trigger switch	0: OFF	1: ON
	3	Receive buffer[*2]	0: No data	1: Data stored
	4	Value of `TIMEA` function	0: Nonzero	1: Zero
	5	Value of `TIMEB` function	0: Nonzero	1: Zero
	6	Value of `TIMEC` function	0: Nonzero	1: Zero
	7	CS (CTS) signal[*3]	0: OFF or file closed	1: ON
3	2-0	LCD contrast level[*4 *5]	0 to 7 (0: Lowest, 7: Highest)	
4	0	Message version[*5 *6]	0: Japanese	1: English
5	0	Alphabet entry function	0: Disabled	1: Enabled
10h-18Fh	7-0	VRAM[*5 *7]	0: OFF	1: ON

[*1] BHT-BASIC represents the bit order by the exponent of each binary digit in the byte. For example, bit 0 means LSB; bit 7 means MSB.

[*2] This status is produced by ORing the receive buffer of the optical interface and that of the direct-connect interface. If either of these buffers has data, therefore, this bit goes ON (1).

[*3] During the direct-connect interface operation, a user program can regard RD signal as CS signal, provided that the returned value of CS should be specified by *`RS/CS`* control parameter in the `OPEN "COM:"` statement as listed below.

`OPEN "COM:"` statement	Returned value of CS (CTS)
`OPEN "COM:,,,,0"`	Always 1
`OPEN "COM:,,,,1"`	Always 1
`OPEN "COM:,,,,2"`	1 if RD signal is High.
`OPEN "COM:,,,,3"`	1 if RD signal is Low.
`OPEN "COM:,,,,4"`	Depends upon the RD signal state.

If the communications device file is closed, the BHT-3000 returns the value 0.

[*4] Lower three bits (bit 2 to bit 0) in this byte represent the contrast level of the LCD in 000 to 111 in binary notation or in 0 to 7 in decimal notation. 0 means the lowest contrast; 7 means the highest.

[*5] The LCD contrast, message version (English/Japanese), and VRAM should not be monitored by using a `WAIT` statement. These status may not change while a user program monitors them by this statement. The `WAIT` statement used for this purpose may cause the program to enter an infinite loop.

[*6] In System Mode, the message version appears as Eng or Jpn on the LCD.

[*7] An 8-bit binary pattern (bits 7 to 0) on the input ports 10h to 18Fh (which read VRAM) represents a basic dot pattern column of the LCD. Bit value 1 means a black dot. The port number gives the dot column address.

■ Output Ports

A user program can control the hardware through the output ports by using the `OUT` statement. BHT-BASIC defines each of these ports as a byte. The table below lists the output ports and their controlling function in the BHT.

Port No.	Bit assignment [*1]	Controls the following:		
1	0	Reading confirmation LED (red)[*2]	0: OFF	1: ON
	1	Reading confirmation LED (green)[*2]	0: OFF	1: ON
3	2-0	LCD contrast level[*3]	0 to 7 (0: Lowest, 7: Highest)	
4	0	Message version	0: Japanese	1: English
5	0	Alphabet entry function	0: Disable	1: Enable
6	7-0	Sleep timer[*4]	0 to 255	
10h-18Fh	7-0	VRAM[*5]	0: OFF	1: ON

[*1] BHT-BASIC represents the bit order by the exponent of each binary digit in the byte. For example, bit 0 means LSB; bit 7 means MSB.

[*2] The reading confirmation LED is controllable only when the bar code device file is closed. If the file is opened, the `OUT` statement will be ignored.

[*3] Lower three bits (bit 2 to bit 0) in this byte control the contrast level of the LCD in 000 to 111 in binary notation or in 0 to 7 in decimal notation. 0 means the lowest contrast; 7 means the highest. Shown below are examples of `OUT` statements.

```
OUT 3,7     ’Contrast is highest
OUT 3,&h07  ’Contrast is highest
```

[*4] The sleep timer feature automatically interrupts program execution if the BHT-3000 receives no input within the specified length of time preset by bits 7 to 0. Shown below are examples of `OUT` statements. Setting 0 to this byte disables the sleep timer feature. (Refer to Chapter 10.)

```
OUT 6,30    ’3 seconds
OUT 6,0     ’No sleep operation
```

[*5] An 8-bit binary pattern (bits 7 to 0) on the output ports 10h to 18Fh (which are stored in the VRAM) represents a basic dot pattern column of the LCD. Bit value 1 means a black dot. The port number gives the dot column address.

D2. BHT-4000

■ Input Ports

A user program can monitor the hardware status through the input ports by using the `WAIT` statement or `INP` function. BHT-BASIC defines each of these ports as a byte. The table below lists the input ports and their monitoring function in the BHT.

Port No.	Bit assignment [*1]	Monitors the following:		
0	0	Keyboard buffer	0: No data	1: Data stored
	1	Barcode buffer	0: No data	1: Data stored
	2	Trigger switch	0: OFF	1: ON
	3	Receive buffer	0: No data	1: Data stored
	4	Value of `TIMEA` function	0: Nonzero	1: Zero
	5	Value of `TIMEB` function	0: Nonzero	1: Zero
	6	Value of `TIMEC` function	0: Nonzero	1: Zero
	7	CS (CTS) signal[*2]	0: OFF or file closed	1: ON
3	2-0	LCD contrast level[*3 *4]	0 to 7 (0: Lowest, 7: Highest)	
4	0	Message version[*4 *5]	0: Japanese	1: English
5	0	Alphabet entry function	0: Disabled	1: Enabled
Dh	0	ER signal[*4*6]	0: OFF	1: ON
Dh	4	CD signal[*6]	0: OFF	1: ON
Eh	0	System status indication[*4*7]	0: OFF	1: ON
Fh	7-0	Re-read prevention enabled time[*8]	0 to 255	
10h-64Fh	7-0	VRAM[*4 *9]	0: OFF	1: ON

[*1] BHT-BASIC represents the bit order by the exponent of each binary digit in the byte. For example, bit 0 means LSB; bit 7 means MSB.

[*2] During the optical interface operation and direct-connect interface operation, a user program can monitor CS (CTS) signal. If CS signal is received, bit 7 of this byte goes ON (1).

If the communications device file is closed, the BHT-4000 returns the value 0.

[*3] Lower three bits (bit 2 to bit 0) in this byte represent the contrast level of the LCD in 000 to 111 in binary notation or in 0 to 7 in decimal notation. 0 means the lowest contrast; 7 means the highest.

[*4] The LCD contrast, message version (English/Japanese), ER signal, system status indication, and VRAM should not be monitored by using a `WAIT` statement. These status may not change while a user program monitors them by this statement. The `WAIT` statement used for this purpose may cause the program to enter an infinite loop.

[*5] In System Mode, the message version appears as English or Japanese on the LCD.

[*6] The ER and CD signals are supported on the direct-connect interface only. If the communications device file is closed, the BHT-4000 returns the value 0.

[*7] The BHT-4000 can display the system status on the bottom line of the LCD. If the system status is displayed, the BHT-4000 returns the value 1; if not, it returns the value 0. For the system status indication, refer to Chapter 7, Subsection 7.1.7.

*8 The BHT-4000 returns the re-read prevention enabled time length in units of 100 ms. If the returned value is zero (0), it means that the re-read prevention is permanently enabled so that the BHT-4000 does not read same bar codes in succession.

*9 An 8-bit binary pattern (bits 7 to 0) on the input ports 10h to 64Fh (which read VRAM) represents a basic dot pattern column of the LCD. Bit value 1 means a black dot. The port number gives the dot column address.

■ Output Ports

A user program can control the hardware through the output ports by using the OUT statement. BHT-BASIC defines each of these ports as a byte. The table below lists the output ports and their controlling function in the BHT.

Port No.	Bit assignment *1	Controls the following:		
1	0	Reading confirmation LED (red)*2	0: OFF	1: ON
	1	Reading confirmation LED (green)*2	0: OFF	1: ON
2	0	RS (RTS) signal*3	0: OFF (Low)	1: ON (High)
3	2-0	LCD contrast level*4	0 to 7 (0: Lowest, 7: Highest)	
4	0	Message version	0: Japanese	1: English
5	0	Alphabet entry function	0: Disable	1: Enable
6	7-0	Sleep timer*5	0 to 255	
Dh	0	ER signal*6	0: OFF	1: ON
Eh	0	System status indication*7	0: OFF	1: ON
Fh	7-0	Re-read prevention enabled time*8	0 to 255	
10h-64Fh	7-0	VRAM*9	0: OFF	1: ON

*1 BHT-BASIC represents the bit order by the exponent of each binary digit in the byte. For example, bit 0 means LSB; bit 7 means MSB.

*2 The reading confirmation LED is controllable only when the bar code device file is closed. If the file is opened, the OUT statement will be ignored.

*3 The RS (RTS) signal is controllable when the communications device file is opened. If the file is closed, this signal specification will be ignored.

*4 Lower three bits (bit 2 to bit 0) in this byte control the contrast level of the LCD in 000 to 111 in binary notation or in 0 to 7 in decimal notation. 0 means the lowest contrast; 7 means the highest. Shown below are examples of OUT statements.

```
OUT 3,7     'Contrast is highest
OUT 3,&h07  'Contrast is highest
```

*5 The sleep time feature automatically interrupts program execution if no event takes place within the specified length of time preset by bit 7 to 0. Shown below are examples of OUT statements. Setting 0 to this byte disables the sleep timer feature. (Refer to Chapter 10.)

```
OUT 6,30    '3 seconds
OUT 6,0     'No sleep operation
```

*6 Available on the direct-connect interface. If the communications device file is closed, this specification will be ignored.

*7 The BHT-4000 may display the system status on the bottom line of the LCD. To display the system status, set 1 to this port; to erase it, set 0. For the system status indication, refer to Chapter 7, Subsection 7.1.7.

*8 This byte sets the re-read prevention enabled time length in units of 100 ms. Specification of zero (0) permanently enables the re-read prevention so that the BHT-4000 does not read same bar codes in succession.

*9 An 8-bit binary pattern (bits 7 to 0) on the output ports 10h to 64Fh (which are stored in the VRAM) represents a basic dot pattern column of the LCD. Bit value 1 means a black dot. The port number gives the dot column address.

If you send graphic data to the VRAM area (assigned to the bottom line of the LCD) by using the `OUT` statement when the system status is displayed on the LCD, the sent data will be written into that VRAM area but cannot be displayed on the bottom line of the LCD.

D3. BHT-5000

■ Input Ports

A user program can monitor the hardware status through the input ports by using the `WAIT` statement or `INP` function. BHT-BASIC defines each of these ports as a byte. The table below lists the input ports and their monitoring function in the BHT.

Port No.	Bit assignment [*1]	Monitors the following:		
0	0	Keyboard buffer	0: No data	1: Data stored
	1	Barcode buffer	0: No data	1: Data stored
	2	Trigger switch	0: OFF	1: ON
	3	Receive buffer	0: No data	1: Data stored
	4	Value of `TIMEA` function	0: Nonzero	1: Zero
	5	Value of `TIMEB` function	0: Nonzero	1: Zero
	6	Value of `TIMEC` function	0: Nonzero	1: Zero
	7	CS (CTS) signal[*2]	0: OFF or file closed	1: ON
3	2-0	LCD contrast level[*3 *4]	0 to 7 (0: Lowest, 7: Highest)	
4	0	Message version[*4 *5]	0: Japanese	1: English
5	0	Alphabet entry function	0: Disabled	1: Enabled
8	0	Wakeup function	0: Deactivated	1: Activated
	1	Initiation of BHT[*6]	0: Initiated by the power key	1: Initiated by the wakeup function
	2	`TIME$` function	0: System time selected	1: Wakeup time selected
	3	Wakeup time	0: Not set	1: Set
Eh	0	System status indication[*4*7]	0: OFF	1: ON
Fh	7-0	Re-read prevention enabled time[*8]	0 to 255	
10h-40Fh	7-0	VRAM[*4 *9]	0: OFF	1: ON
6010h	7-0	Battery voltage level[*10]	0 to 255	
6011h	0	Battery type	0: Rechargeable battery cartridge	1: Dry battery cartridge
6040h	0	Magic key 1	0: Released	1: Held down
	1	Magic key 2	0: Released	1: Held down
6050h	0	Keyboard type	0: 26-key pad	1: 32-key pad
6060h	7-0	Communications protocol[*11]	0: BHT-protocol	1: Multilink protocol
6061h	7-0	ID (lower byte)[*12]	0 to 255	
6062h	7-0	ID (upper byte)[*12]	0 to 255	

[*1] BHT-BASIC represents the bit order by the exponent of each binary digit in the byte. For example, bit 0 means LSB; bit 7 means MSB.

[*2] During the optical interface operation, a user program can monitor CS (CTS) signal. If CS signal is received, bit 7 of this byte goes ON (1).

During the direct-connect interface operation, a user program can regard RD signal as CS signal, provided that the returned value of CS should be specified by *RS/CS* control parameter in the `OPEN "COM:"` statement as listed below.

`OPEN "COM:"` statement	Returned value of CS (CTS)
`OPEN "COM:,,,,0"`	Always 1
`OPEN "COM:,,,,1"`	Always 1
`OPEN "COM:,,,,2"`	1 if RD signal is High.
`OPEN "COM:,,,,3"`	1 if RD signal is Low.
`OPEN "COM:,,,,4"`	Depends upon the RD signal state.

If the communications device file is closed, the BHT-5000 returns the value 0.

*3 Lower three bits (bit 2 to bit 0) in this byte represent the contrast level of the LCD in 000 to 111 in binary notation or in 0 to 7 in decimal notation. 0 means the lowest contrast; 7 means the highest.

*4 The LCD contrast, message version (English/Japanese), system status indication, and VRAM should not be monitored by using a `WAIT` statement. These status may not change while a user program monitors them by this statement. The `WAIT` statement used for this purpose may cause the program to enter an infinite loop.

*5 In System Mode, the message version appears as English or Japanese on the LCD.

*6 If the BHT-5000 is initiated by the wakeup function, this bit goes ON (1).

*7 The BHT-5000 can display the system status on the bottom line of the LCD. If the system status is displayed, the BHT-5000 returns the value 1; if not, it returns the value 0. For the system status indication, refer to Chapter 7, Subsection 7.1.7.

*8 The BHT-5000 returns the re-read prevention enabled time length in units of 100 ms. If the returned value is zero (0), it means that the re-read prevention is permanently enabled so that the BHT-5000 does not read same bar codes in succession.

*9 An 8-bit binary pattern (bits 7 to 0) on the input ports 10h to 40Fh (which read VRAM) represents a basic dot pattern column of the LCD. Bit value 1 means a black dot. The port number gives the dot column address.

*10 A user program returns the A/D converted value (0 to 255) of the battery voltage level (0 to 5V). The returned value is an instantaneous value when data on the input port is read. The voltage level varies depending upon the BHT-5000 operation and it is not in proportion to the battery capacity, so use this voltage level as a reference value.

*11 A user program returns the communications protocol type used for file transmission with the `XFILE` statement.

*12 A user program returns the BHT's ID number which is required for the use of the multilink protocol. The ID number is expressed by two bytes: lower byte on port 6061h and upper byte on port 6062h. The range of the returned value is from 1 to FFFFh. If the ID number is 1234h, for example, the value on 6061h is 34h and that on 6062h is 12h.

■ Output Ports

A user program can control the hardware through the output ports by using the OUT statement. BHT-BASIC defines each of these ports as a byte. The table below lists the output ports and their controlling function in the BHT.

Port No.	Bit assignment [*1]	Controls the following:		
1	0	Reading confirmation LED (red)[*2]	0: OFF	1: ON
	1	Reading confirmation LED (green)[*2]	0: OFF	1: ON
2	0	RS (RTS) signal[*3]	0: OFF (Low)	1: ON (High)
3	2-0	LCD contrast level[*4]	0 to 7 (0: Lowest, 7: Highest)	
4	0	Message version	0: Japanese	1: English
5	0	Alphabet entry function	0: Disable	1: Enable
6	7-0	Sleep timer[*5]	0 to 255	
8	0	Wakeup function[*6]	0: Deactivate	1: Activate
	2	TIME$ function[*7]	0: Select the system time	1: Select the wakeup time
Eh	0	System status indication[*8]	0: OFF	1: ON
Fh	7-0	Re-read prevention enabled time[*9]	0 to 255	
10h-40Fh	7-0	VRAM[*10]	0: OFF	1: ON
6000h	0	Initiation of System Mode[*11]	0: Do not initiate	1: Initiate
6020h	0	LCD backlight[*12]	0: Turns OFF	1: Turns ON
6021h	7-0	LCD backlight ON-duration[*12]	0 to 255	
6030h	7-0	Effective held-down time of power key[*13]	1 to 255	
6060h	7-0	Communications protocol[*14]	0: BHT-protocol	1: Multilink protocol
6061h	7-0	ID (lower byte)[*15]	0 to 255	
6062h	7-0	ID (upper byte)[*15]	0 to 255	

*1 BHT-BASIC represents the bit order by the exponent of each binary digit in the byte. For example, bit 0 means LSB; bit 7 means MSB.

*2 The reading confirmation LED is controllable only when the bar code device file is closed. If the file is opened, the OUT statement will be ignored.

If you have set the confirmation LED to OFF in the OPEN "BAR:" statement, a user program can control the reading confirmation LED although the bar code device file is opened.

*3 The RS (RTS) signal is controllable on the optical interface.

*4 Lower three bits (bit 2 to bit 0) in this byte control the contrast level of the LCD in 000 to 111 in binary notation or in 0 to 7 in decimal notation. 0 means the lowest contrast; 7 means the highest. Shown below are examples of OUT statements.

```
OUT 3,7     'Contrast is highest
OUT 3,&h07  'Contrast is highest
```

*5 The sleep timer feature automatically interrupts program execution if no event takes place within the specified length of time preset by bit 7 to 0. Shown below are examples of OUT statements. Setting 0 to this byte disables the sleep timer feature. (Refer to Chapter 10.)

```
OUT 6,30    '3 seconds
OUT 6,0     'No sleep operation
```

*6 To activate the wakeup function, set 1 to this bit; to deactivate it, set 0.

*7 To make the TIME$ function return or set the system time, set 0 to this bit; to make the TIME$ function return or set the wakeup time, set 1.

Execution of the TIME$ function after selection of the wakeup time will automatically reset this bit to zero.

*8 The BHT-5000 may display the system status on the bottom line of the LCD. To display the system status, set 1 to this port; to erase it, set 0. For the system status indication, refer to Chapter 7, Subsection 7.1.7.

*9 This byte sets the re-read prevention enabled time length in units of 100 ms. Specification of zero (0) permanently enables the re-read prevention so that the BHT-5000 does not read same bar codes in succession. The default is 10 (1 second).

*10 An 8-bit binary pattern (bits 7 to 0) on the output ports 10h to 40Fh (which are stored in the VRAM) represents a basic dot pattern column of the LCD. Bit value 1 means a black dot. The port number gives the dot column address.

If you send graphic data to the VRAM area (assigned to the bottom line of the LCD) by using the OUT statement when the system status is displayed on the LCD, the sent data will be written into that VRAM area but cannot be displayed on the bottom line of the LCD.

*11 Refer to Appendix H, "[3] Program file named APLINT.PD3."

*12 If the backlight function is activated with the OUT statement, the specification by the KEY statement will be ignored. For details, refer to Chapter 13.

If you set 0 to the ON-duration (6021h), the backlight will not come on; if you set 255, it will be kept on.

*13 You can set the held-down time of the power key required for powering off the BHT-5000. The setting range is from 0.1 to 25.5 seconds in increments of 0.1 second. The default is 5 (0.5 second).

*14 You can set the communications protocol type for transmitting files with the XFILE statement. To transmit files between the host computer and more than one BHT-5000 (placed on the multilinked CU-5003s), set 1 (multilink protocol) to this port. The file transmission by using the multilink protocol requires also Multilink Transfer Utility (MLTU3.EXE) to be run in the host computer, Multilink Protocol System (MLTU3.EX3) to be run in the BHT-5000, and the CU-5003(s).

If Multilink Protocol System (MLTU3.EX3) has not been downloaded to the BHT-5000, the BHT-protocol will be used instead of the multilink protocol even if this port is set to 1 (multilink protocol).

*15 You may set the BHT's ID number to be used for the multilink protocol. The ID number is expressed by two bytes: lower byte on port 6061h and upper byte on port 6062h. The setting range is from 1 to FFFFh. To set the ID number to 1234h, write as follows:

```
OUT &h6061h,&h34  'Sets 34h to the lower byte of the ID
OUT &h6062h,&h12  'Sets 12h to the upper byte of the ID
```

D4. BHT-6000/BHT-6500

■ Input Ports

A user program can monitor the hardware status through the input ports by using the `WAIT` statement or `INP` function. BHT-BASIC defines each of these ports as a byte. The table below lists the input ports and their monitoring function in the BHT.

Port No.	Bit assignment [*1]	Monitors the following:		
0	0	Keyboard buffer	0: No data	1: Data stored
	1	Barcode buffer	0: No data	1: Data stored
	2	Trigger switch[*2]	0: OFF	1: ON
	3	Receive buffer	0: No data	1: Data stored
	4	Value of `TIMEA` function	0: Nonzero	1: Zero
	5	Value of `TIMEB` function	0: Nonzero	1: Zero
	6	Value of `TIMEC` function	0: Nonzero	1: Zero
	7	CS (CTS) signal[*3]	0: OFF or file closed	1: ON
3	3-0	LCD contrast level[*4 *5]	0 to 11 (0: Lowest, 11: Highest)	
4	0	Message version[*5 *6]	0: Japanese	1: English
5	0	Alphabet entry function	0: Disabled	1: Enabled
8	0	Wakeup function	0: Deactivated	1: Activated
	1	Initiation of BHT[*7]	0: Initiated by the power key	1: Initiated by the wakeup function
	2	`TIME$` function	0: System time selected	1: Wakeup time selected
	3	Wakeup time	0: Not set	1: Set
Eh	0	System status indication[*5*8]	0: OFF	1: ON
Fh	7-0	Re-read prevention enabled time[*9]	0 to 255	
10h-24Fh	7-0	VRAM[*5 *10]	0: OFF	1: ON
6010h	7-0	Battery voltage level[*11]	0 to 255	
6011h	0	Battery type	0: Battery cartridge	1: Dry batteries
6040h	0	Magic key 1	0: Released	1: Held down
	1	Magic key 2	0: Released	1: Held down
	2[*12]	Magic key 3	0: Released	1: Held down
	3[*12]	Magic key 4	0: Released	1: Held down
6060h	7-0	Communications protocol[*13]	0: BHT-protocol	2: BHT-Ir protocol
6061h	7-0	ID (lower byte)[*14]	0 to 255	
6062h	7-0	ID (upper byte)[*14]	0 to 255	
6070h	0	Output pulse width of IR beam	0: 1.63 µs	1: 3/16 bit time
6080h	0	Display font size[*15]	0: Standard-size	1: Small-size
6090h[*12]	0	Beeper	0: Deactivated	1: Activated
	1	Vibrator	0: Deactivated	1: Activated

*1 BHT-BASIC represents the bit order by the exponent of each binary digit in the byte. For example, bit 0 means LSB; bit 7 means MSB.

*2 Only when the trigger switch function is assigned to any of the magic keys, a user program returns the ON/OFF state of the switch.

*3 During the direct-connect interface operation, a user program can regard RD signal as CS signal, provided that the returned value of CS should be specified by *`RS/CS`* control parameter in the `OPEN "COM:"` statement as listed below.

`OPEN "COM:"` statement	Returned value of CS (CTS)
`OPEN "COM:,,,,0"`	Always 1
`OPEN "COM:,,,,1"`	Always 1
`OPEN "COM:,,,,2"`	1 if RD signal is High.
`OPEN "COM:,,,,3"`	1 if RD signal is Low.
`OPEN "COM:,,,,4"`	Depends upon the RD signal state.

If the direct-connect interface is closed, the BHT-6000/BHT-6500 returns the value 0.

*4 Lower four bits (bit 3 to bit 0) in this byte represent the contrast level of the LCD in 0000 to 1011 in binary notation or in 0 to 11 in decimal notation. 0 means the lowest contrast; 11 means the highest.

*5 The LCD contrast, message version (English/Japanese), system status indication, and VRAM should not be monitored by using a `WAIT` statement. These status may not change while a user program monitors them by this statement. The `WAIT` statement used for this purpose may cause the program to enter an infinite loop.

*6 In System Mode, the message version appears as English or Japanese on the LCD.

*7 If the BHT-6000/BHT-6500 is initiated by the wakeup function, this bit goes ON (1).

*8 The BHT-6000/BHT-6500 can display the system status on the bottom line of the LCD. If the system status is displayed, the BHT-6000/BHT-6500 returns the value 1; if not, it returns the value 0. For the system status indication, refer to Chapter 7, Subsection 7.1.7.

*9 The BHT-6000/BHT-6500 returns the re-read prevention enabled time length in units of 100 ms. If the returned value is zero (0), it means that the re-read prevention is permanently enabled so that the BHT-6000/BHT-6500 does not read same bar codes in succession.

*10 An 8-bit binary pattern (bits 7 to 0) on the input ports 10h to 24Fh (which read VRAM) represents a basic dot pattern column of the LCD. Bit value 1 means a black dot. The port number gives the dot column address.

*11 A user program returns the A/D converted value (0 to 255) of the battery voltage level (0 to 5V in the BHT-6000 and 0 to 3.5V in the BHT-6500). The returned value is an instantaneous value when data on the input port is read. The voltage level varies depending upon the BHT-6000/BHT-6500 operation and it is not in proportion to the battery capacity, so use this voltage level as a reference value.

*12 Supported by the BHT-6500 only.

*13 A user program returns the communications protocol type used for file transmission with the `XFILE` statement. For details about the communications protocol, refer to the "BHT-6000 User's Manual" or "BHT-6500 User's Manual."

[*14] A user program returns the BHT's ID number which is required for the use of the BHT-Ir protocol. The ID number is expressed by two bytes: lower byte on port 6061h and upper byte on port 6062h. The range of the returned value is from 1 to FFFFh. If the ID number is 1234h, for example, the value on 6061h is 34h and that on 6062h is 12h.

[*15] If the value of this bit is 0 (standard-size), characters will be displayed as follows:

		(W) x (H)
Single-byte ANK mode		6 dots x 8 dots
Two-byte Kanji mode	Full-width	16 dots x 16 dots
	Half-width	8 dots x 16 dots

If the value of this bit is 1 (small-size), characters will be displayed as follows:

		(W) x (H)
Single-byte ANK mode		6 dots x 6 dots
Two-byte Kanji mode	Full-width	12 dots x 12 dots
	Half-width	6 dots x 12 dots

■ Output Ports

A user program can control the hardware through the output ports by using the OUT statement. BHT-BASIC defines each of these ports as a byte. The table below lists the output ports and their controlling function in the BHT.

Port No.	Bit assignment [*1]	Controls the following:		
1	0 1	Reading confirmation LED (red)[*2] Reading confirmation LED (green)[*2]	0: OFF 0: OFF	1: ON 1: ON
3	3-0	LCD contrast level[*3]	0 to 11 (0: Lowest, 11: Highest)	
4	0	Message version	0: Japanese	1: English
5	0	Alphabet entry function	0: Disable	1: Enable
6	7-0	Sleep timer[*4]	0 to 255	
8	0 2	Wakeup function[*5] TIME$ function[*6]	0: Deactivate 0: Select the system time	1: Activate 1: Select the wakeup time
Eh	0	System status indication[*7]	0: OFF	1: ON
Fh	7-0	Re-read prevention enabled time[*8]	0 to 255	
10h-24Fh	7-0	VRAM[*9]	0: OFF	1: ON
6000h	0	Initiation of System Mode[*10]	0: Do not initiate	1: Initiate
6020h 6021h	0 7-0	LCD backlight[*11] LCD backlight ON-duration[*11]	0: Turns OFF 0 to 255	1: Turns ON
6030h	7-0	Effective held-down time of power key[*12]	1 to 255	
6060h	7-0	Communications protocol[*13]	0: BHT-protocol	2: BHT-Ir protocol
6061h 6062h	7-0 7-0	ID (lower byte)[*14] ID (upper byte)[*14]	0 to 255 0 to 255	
6070h	0	Output pulse width of IR beam[*15]	0: 1.63 µs	1: 3/16 bit time
6080h	0	Display font size[*16]	0: Standard-size	1: Small-size
6090h[*17]	0 1	Beeper Vibrator	0: Deactivate 0: Deactivate	1: Activate 1: Activate

*1 BHT-BASIC represents the bit order by the exponent of each binary digit in the byte. For example, bit 0 means LSB; bit 7 means MSB.

*2 The reading confirmation LED is controllable only when the bar code device file is closed. If the file is opened, the `OUT` statement will be ignored.

If you have set the confirmation LED to OFF in the `OPEN "BAR:"` statement a user program can control the reading confirmation LED although the bar code device file is opened.

*3 Lower four bits (bit 3 to bit 0) in this byte control the contrast level of the LCD in 0000 to 1011 in binary notation or in 0 to 11 in decimal notation. 0 means the lowest contrast; 11 means the highest. Shown below are examples of `OUT` statements.

```
OUT 3,11    ’Contrast is highest
OUT 3,&h0B  ’Contrast is highest
```

*4 The sleep timer feature automatically interrupts program execution if no event takes place within the specified length of time preset by bit 7 to 0. Shown below are examples of `OUT` statements. Setting 0 to this byte disables the sleep timer feature. (Refer to Chapter 10.)

```
OUT 6,30    ’3 seconds
OUT 6,0     ’No sleep operation
```

*5 To activate the wakeup function, set 1 to this bit; to deactivate it, set 0.

*6 To make the `TIME$` function return or set the system time, set 0 to this bit; to make the `TIME$` function return or set the wakeup time, set 1.

Execution of the `TIME$` function after selection of the wakeup time will automatically reset this bit to zero.

*7 The BHT-6000/BHT6500 may display the system status on the bottom line of the LCD. To display the system status, set 1 to this port; to erase it, set 0. For the system status indication, refer to Chapter 7, Subsection 7.1.7.

*8 This byte sets the re-read prevention enabled time length in units of 100 ms. Specification of zero (0) permanently enables the re-read prevention so that the BHT-6000/BHT6500 does not read same bar codes in succession. The default is 10 (1 second).

*9 An 8-bit binary pattern (bits 7 to 0) on the output ports 10h to 24Fh (which are stored in the VRAM) represents a basic dot pattern column of the LCD. Bit value 1 means a black dot. The port number gives the dot column address.

If you send graphic data to the VRAM area (assigned to the bottom line of the LCD) by using the `OUT` statement when the system status is displayed on the LCD, the sent data will be written into that VRAM area but cannot be displayed on the bottom line of the LCD.

*10 Refer to Appendix H, "[3] Program file named APLINT.PD3."

*11 If the backlight function is activated with the `OUT` statement, the specification by the `KEY` statement will be ignored. For details, refer to Chapter 13.

If you set 0 to the ON-duration (6021h), the backlight will not come on; if you set 255, it will be kept on.

*12 You can set the held-down time of the power key required for powering off the BHT-6000/BHT6500. The setting range is from 0.1 to 25.5 seconds in increments of 0.1 second. The default is 5 (0.5 second).

[13] You can set the communications protocol type for transmitting files with the `XFILE` statement.

[14] You may set the BHT's ID number to be used for the BHT-Ir protocol. The ID number is expressed by two bytes: lower byte on port 6061h and upper byte on port 6062h. The setting range is from 1 to FFFFh. To set the ID number to 1234h, write as follows:

```
OUT &h6061h,&h34  'Sets 34h to the lower byte of the ID
OUT &h6062h,&h12  'Sets 12h to the upper byte of the ID
```

[15] For data transmission via the optical interface, this bit sets the output pulse width of IR beam in accordance with the IrDA physical layer (IrDA-SIR 1.0). The default width is 1.63 µs.

[16] If you set 0 (standard-size) to this bit, characters will be displayed as follows:

		(W) x (H)
Single-byte ANK mode		6 dots x 8 dots
Two-byte Kanji mode	Full-width	16 dots x 16 dots
	Half-width	8 dots x 16 dots

If you set 1 (small-size) to this bit, characters will be displayed as follows:

		(W) x (H)
Single-byte ANK mode		6 dots x 6 dots
Two-byte Kanji mode	Full-width	12 dots x 12 dots
	Half-width	6 dots x 12 dots

[17] Supported by the BHT-6500 only. If you set 0 (Deactivates) to both bits 0 and 1, only the beeper will work.

D5. BHT-7000/BHT-7500

■ Input Ports

A user program can monitor the hardware status through the input ports by using the `WAIT` statement or `INP` function. BHT-BASIC defines each of these ports as a byte. The table below lists the input ports and their monitoring function in the BHT.

Port No.	Bit assignment [*1]	Monitors the following:		
0	0	Keyboard buffer	0: No data	1: Data stored
	1	Barcode buffer	0: No data	1: Data stored
	2	Trigger switch[*2]	0: OFF	1: ON
	3	Receive buffer	0: No data	1: Data stored
	4	Value of `TIMEA` function	0: Nonzero	1: Zero
	5	Value of `TIMEB` function	0: Nonzero	1: Zero
	6	Value of `TIMEC` function	0: Nonzero	1: Zero
	7	CS (CTS) signal[*3]	0: OFF or file closed	1: ON
3	2-0	LCD contrast level[*4]	0 to 7 (0: Lowest, 7: Highest)	
4	0	Message version[*5]	0: Japanese	1: English
5	0	Alphabet entry function	0: Disabled	1: Enabled
8	0	Wakeup function	0: Deactivated	1: Activated
	1	Initiation of BHT[*6]	0: Initiated by the power key	1: Initiated by the wakeup function
	2	`TIME$` function	0: System time selected	1: Wakeup time selected
	3	Wakeup time	0: Not set	1: Set
Fh	7-0	Re-read prevention enabled time[*7]	0 to 255	
10h-40Fh	7-0	BHT-7000 VRAM[*8]	0: OFF	1: ON
10h-C8Fh	7-0	BHT-7500 VRAM[*8]	0: OFF	1: ON
6010h	7-0	Battery voltage level[*9]	0 to 255	
6011h	0	Battery type	0: Rechargeable battery cartridge	1: Dry battery cartridge
6012h	0	BHT on/off the CU[*10]	0: Off the CU 1: On the CU 2: Loaded with dry battery cartridge	
6040h	0	Magic key 1	0: Released	1: Held down
	1	Magic key 2	0: Released	1: Held down
	2	Magic key 3	0: Released	1: Held down
	3	Magic key 4	0: Released	1: Held down
6060h	7-0	Communications protocol[*11]	0: BHT-protocol	2: BHT-Ir protocol
6061h	7-0	ID (lower byte)[*12]	0 to 255	
6062h	7-0	ID (upper byte)[*12]	0 to 255	
6070h	0	Output pulse width of IR beam[*13]	1: 3/16 bit time	
6080h	0	Display font size[*14]	0: Standard-size	1: Small-size

Port No.	Bit assignment *1	Monitors the following:		
6090h	0	Beeper	0: Deactivated	1: Activated
	1	Vibrator	0: Deactivated	1: Activated
60B0h*15	7-0	Key entry system	0: Numeric entry	1: Alphanumeric entry
60B1h*15	7-0	Key entry mode	0: Numeric	1: Alphabet
60C0h	7-0	Beeper volume*16	0 to 3	
60E0h	7-0	Drive size to be defragmented (lower byte)*17		0 to 255
60E1h	7-0	Drive size to be defragmented (upper byte)*17		0 to 255
60F0h	7-0	Remote wakeup function*18	0: Deactivated	1: Activated
60F1h	7-0	Transmission speed for remote wakeup*19	1: 9600 bps 3: 38400 bps 5: 115200 bps	2: 19200 bps 4: 57600 bps
60F2h	0	Execution record of remote wakeup*20	1: Woke up remotely	
	1	Termination of remote wakeup*21	1: Terminated normally	
60F3h	7-0	Timeout for remote wakeup*22	1 to 255 (seconds)	

*1 BHT-BASIC represents the bit order by the exponent of each binary digit in the byte. For example, bit 0 means LSB; bit 7 means MSB.

*2 Only when the trigger switch function is assigned to either of the magic keys, a user program returns the ON/OFF state of the switch.

*3 During the direct-connect interface operation, a user program can regard RD signal as CS signal, provided that the returned value of CS should be specified by *RS/CS* control parameter in the `OPEN "COM:"` statement as listed below.

`OPEN "COM:"` statement	Returned value of CS (CTS)
`OPEN "COM:,,,,0"`	Always 1
`OPEN "COM:,,,,1"`	Always 1
`OPEN "COM:,,,,2"`	1 if RD signal is High.
`OPEN "COM:,,,,3"`	1 if RD signal is Low.
`OPEN "COM:,,,,4"`	Depends upon the RD signal state.

If the direct-connect interface is closed, the BHT-7000/BHT-7500 returns the value 0.

*4 Lower three bits (bit 2 to bit 0) in this byte represent the contrast level of the LCD in 000 to 111 in binary notation or in 0 to 7 in decimal notation. 0 means the lowest contrast; 7 means the highest.

*5 In System Mode, the message version appears as English or Japanese on the LCD.

*6 If the BHT-7000/BHT-7500 is initiated by the wakeup function, this bit goes ON (1).

*7 The BHT-7000/BHT-7500 returns the re-read prevention enabled time length in units of 100 ms. If the returned value is zero (0), it means that the re-read prevention is permanently enabled so that the BHT-7000/BHT-7500 does not read same bar codes in succession.

*8 An 8-bit binary pattern (bits 7 to 0) on the input ports (which read VRAM) 10h to 40Fh in the BHT-7000 or 10h to C8Fh in the BHT-7500 represents a basic dot pattern column of the LCD. Bit value 1 means a black dot. The port number gives the dot column address.

On input ports BE0h to C8Fh, which represents the bottom line of the LCD, is a 7-bit binary pattern (bits 6 to 0) only. If "1" is set to bit 7 for output, bit 7 returns "1"; if "0," bit 7 returns "0."

*9 A user program returns the A/D converted value (0 to 255) of the battery voltage level (0 to 7V). The returned value is an instantaneous value when data on the input port is read. The voltage level varies depending upon the BHT-7000/BHT-7500 operation and it is not in proportion to the battery capacity, so use this voltage level as a reference value.

*10 If the BHT is placed on the CU and is ready to be charged (or being charged), "1" will be returned. In this condition, the indicator LED on the BHT is lit in red or green showing the charging state.

In either of the following cases, "0" will be returned even if the BHT is placed on the CU:

- No power is supplied to the CU.
- The BHT cannot be recognized as being placed on the CU due to contact failure of charging terminals.

*11 A user program returns the communications protocol type used for file transmission with the `XFILE` statement. For details about the communications protocol, refer to the "BHT-7000 User's Manual" or "BHT-7500 User's Manual."

*12 A user program returns the BHT's ID number which is required for the use of the BHT-Ir protocol. The ID number is expressed by two bytes: lower byte on port 6061h and upper byte on port 6062h. The range of the returned value is from 1 to FFFFh. If the ID number is 1234h, for example, the value on 6061h is 34h and that on 6062h is 12h.

*13 Fixed to 3/16 bit time.

*14 If the value of this bit is 0 (standard-size), characters will be displayed as follows:

		(W) x (H)
Single-byte ANK mode		6 dots x 8 dots
Two-byte Kanji mode	Full-width	16 dots x 16 dots
	Half-width	8 dots x 16 dots

If the value of this bit is 1 (small-size), characters will be displayed as follows:

		(W) x (H)
Single-byte ANK mode		6 dots x 6 dots
Two-byte Kanji mode	Full-width	12 dots x 12 dots
	Half-width	6 dots x 12 dots

*15 Valid only in the BHT-7000 with 26-key pad.

*16 A user program returns the beeper volume level--1 (Low), 2 (Medium), or 3 (High). 0 means no beeping.

[17] A user program returns the currently specified size of the empty area to be defragmented in units of 4 kilobytes. The size is expressed by two bytes: lower byte on port 60E0h and upper byte on port 60E1h. The range of the returned value is from 1 to FFFFh. (The actually allowable maximum value is the size of the empty user area. If a value exceeding the size is returned, it means that the whole empty area is specified to be defragmented.)

If the size is 2048 kilobytes, for example, the value on 60E0h is 00h and that on 60E1h is 02h (2048 kilobytes/4 kilobytes = 512 or 200h). 0 means the whole empty area to be defragmented.

[18] If "0" is returned, the remote wakeup function is deactivated; if "1," the function is activated.

[19] The transmission speed to be applied when activating the remote wakeup will be returned.

[20] If the BHT was woke up remotely at the last powering on, "1" will be returned; if the BHT is initiated from any other means, "0" will be returned.

[21] If a user program executed by the remote wakeup has been terminated with `END`, `POWER OFF`, or `POWER 0` statement, then "1" will be returned; in any other cases, "0" will be returned.

[22] A user program returns the timeout length during which the BHT will wait for proper data (specified remote wakeup character string) after receiving any data via the CU from the host.

■ Output Ports

A user program can control the hardware through the output ports by using the `OUT` statement. BHT-BASIC defines each of these ports as a byte. The table below lists the output ports and their controlling function in the BHT.

Port No.	Bit assignment [1]	Controls the following:		
1	0	Reading confirmation LED (red)[2]	0: OFF	1: ON
	1	Reading confirmation LED (green)[2]	0: OFF	1: ON
3	2-0	LCD contrast level[3]	0 to 7 (0: Lowest, 7: Highest)	
4	0	Message version	0: Japanese	1: English
5	0	Alphabet entry function	0: Disable	1: Enable
6	7-0	Sleep timer[4]	0 to 255	
8	0	Wakeup function[5]	0: Deactivate	1: Activate
	2	`TIME$` function[6]	0: Select the system time	1: Select the wakeup time
Fh	7-0	Re-read prevention enabled time[7]	0 to 255	
10h-40Fh	7-0	BHT-7000 VRAM[8]	0: OFF	1: ON
10h-C8Fh	7-0	BHT-7500 VRAM[8]	0: OFF	1: ON
6000h	0	Initiation of System Mode[9]	0: Do not initiate	1: Initiate
6020h	0	LCD backlight[10]	0: Turn OFF	1: Turn ON
6021h	7-0	LCD backlight ON-duration[10]	0 to 255	
6030h	7-0	Effective held-down time of power key[11]	1 to 255	

Port No.	Bit assignment [1]	Controls the following:		
6060h	7-0	Communications protocol[12]	0: BHT-protocol	2: BHT-Ir protocol
6061h	7-0	ID (lower byte)[13]	0 to 255	
6062h	7-0	ID (upper byte)[13]	0 to 255	
6080h	7-0	Display font size[14]	0: Standard-size	1: Small-size
6090h	0	Beeper[15]	0: Deactivates	1: Activates
	1	Vibrator[15]	0: Deactivates	1: Activates
60B0h[16]	7-0	Key entry system	0: Numeric entry	1: Alphanumeric entry
60B1h[16]	7-0	Key entry mode	0: Numeric	1: Alphabet
60C0h	7-0	Beeper volume[17]	0 to 3	
60D0h	7-0	System modification[18]	0: Power off after modification	
			1: Software-reset after modification	
60E0h	7-0	Drive size to be defragmented (lower byte)[19]		0 to 255
60E1h	7-0	Drive size to be defragmented (upper byte)[19]		0 to 255
60E2h	7-0	Execution of defragmentation[20]	0: Defragments w/o bar graph	
			1: Defragments w/ absolute bar graph	
			2: Defragments w/ relative bar graph	
60F0h	7-0	Remote wakeup function[21]	0: Deactivated	1: Activated
60F1h	7-0	Transmission speed for remote wakeup[22]	1: 9600 bps	2: 19200 bps
			3: 38400 bps	4: 57600 bps
			5: 115200 bps	
60F2h	0	Execution record of remote wakeup	1: Woke up remotely	
	1	Termination of remote wakeup	1: Terminated normally	
60F3h	7-0	Timeout for remote wakeup[23]	1 to 255 (seconds)	

[1] BHT-BASIC represents the bit order by the exponent of each binary digit in the byte. For example, bit 0 means LSB; bit 7 means MSB.

[2] The reading confirmation LED is controllable only when the bar code device file is closed. If the file is opened, the `OUT` statement will be ignored.

If you have set the confirmation LED to OFF in the `OPEN "BAR:"` statement, a user program can control the reading confirmation LED although the bar code device file is opened.

[3] Lower three bits (bit 2 to bit 0) in this byte control the contrast level of the LCD in 000 to 111 in binary notation or in 0 to 7 in decimal notation. 0 means the lowest contrast; 7 means the highest.

```
OUT 3,7     'Contrast is highest
OUT 3,&h07  'Contrast is highest
```

[4] The sleep timer feature automatically interrupts program execution if no event takes place within the specified length of time preset by bit 7 to 0. Shown below are examples of `OUT` statements. Setting 0 to this byte disables the sleep timer feature. (Refer to Chapter 10.)

```
OUT 6,30    '3 seconds
OUT 6,0     'No sleep operation
```

[5] To activate the wakeup function, set 1 to this bit; to deactivate it, set 0.

*6 To make the `TIME$` function return or set the system time, set 0 to this bit; to make the `TIME$` function return or set the wakeup time, set 1.

Execution of the `TIME$` function after selection of the wakeup time will automatically reset this bit to zero.

*7 This byte sets the re-read prevention enabled time length in units of 100 ms. Specification of zero (0) permanently enables the re-read prevention so that the BHT-7000/BHT7500 does not read same bar codes in succession. The default is 10 (1 second).

*8 An 8-bit binary pattern (bits 7 to 0) on the output ports (which are stored in the VRAM) 10h to 40Fh in the BHT-7000 or 10h to C8Fh in the BHT-7500 represents a basic dot pattern column of the LCD. Bit value 1 means a black dot. The port number gives the dot column address.

On input ports BE0h to C8Fh, which represents the bottom line of the LCD, is a 7-bit binary pattern (bits 6 to 0) only. If you set "1" to bit 7, it will be displayed as 1; if "0," it will be as 0.

*9 Refer to Appendix H, "[3] Program file named APLINT.PD3."

*10 If the backlight function is activated with the `OUT` statement, the specification by the `KEY` statement will be ignored. For details, refer to Chapter 13.

If you set 0 to the ON-duration (6021h), the backlight will not come on; if you set 255, it will be kept on.

*11 You can set the held-down time of the power key required for powering off the BHT-7000/ BHT7500. The setting range is from 0.1 to 25.5 seconds in increments of 0.1 second. The default is 5 (0.5 second).

*12 You can set the communications protocol type for transmitting files with the `XFILE` statement.

*13 You may set the BHT's ID number to be used for the BHT-Ir protocol. The ID number is expressed by two bytes: lower byte on port 6061h and upper byte on port 6062h. The setting range is from 1 to FFFFh. To set the ID number to 1234h, write as follows:

```
OUT &h6061h,&h34  ’Sets 34h to the lower byte of the ID
OUT &h6062h,&h12  ’Sets 12h to the upper byte of the ID
```

*14 If you set 0 (standard-size) to this bit, characters will be displayed as follows:

		(W) x (H)
Single-byte ANK mode		6 dots x 8 dots
Two-byte Kanji mode	Full-width	16 dots x 16 dots
	Half-width	8 dots x 16 dots

If you set 1 (small-size) to this bit, characters will be displayed as follows:

		(W) x (H)
Single-byte ANK mode		6 dots x 6 dots
Two-byte Kanji mode	Full-width	12 dots x 12 dots
	Half-width	6 dots x 12 dots

*15 If you set 0 (Deactivates) to both bits 0 and 1, only the beeper will work.

*16 Valid only in the BHT-7000 with 26-key pad.

*17 The beeper volume level may be adjusted to four levels--1 (Low), 2 (Medium), 3 (High), and 0 (OFF).

[*18] To update the BHT system by using an application program, download an update file to the BHT and then execute an `OUT` statement. Updating the system will take approx. 30 seconds. During updating, the BHT power should be kept on. If an execution program has been set, execution of `OUT &H60D0`, 1 may cold-start the application.

[*19] You may specify the size of the empty user area to be defragmented in units of 4 kilobytes. The size is expressed by two bytes: lower byte on port 60E0h and upper byte on port 60E1h. The setting range is from 1 to FFFFh. (The actually allowable maximum value is the size of the empty user area. If you specify a value exceeding the size, the whole empty area will be defragmented.)

To defragment 2048 kilobytes of area, write as follows:

2048 kilobytes/4 kilobytes = 512 (200h), so

```
OUT &h60E0,0    'Sets 00h to the lower byte
OUT &h60E1,2    'Sets 02h to the upper byte
```

If "0" is set, the whole empty user area will be defragmented.

[*20] To defragment the drive, set "0," "1," or "2." Setting "1" or "2" will display an absolute bar graph or relative bar graph indicating the defragmentation progress during drive defragmentation, respectively. The bar graph will disappear after completion of defragmentation and the previous screen will come back.

To defragment the drive while showing a relative bar graph, write as follows:

```
OUT &h60E2,1  'Defragment the drive showing relative bar
graph
```

[*21] To activate the remote wakeup, set "1"; to deactivate, set "0."

[*22] Set the transmission speed to be applied for remote wakeup.

[*23] You may set the timeout length during which the BHT will wait for proper data (specified remote wakeup character string) after receiving any data via the CU from the host.

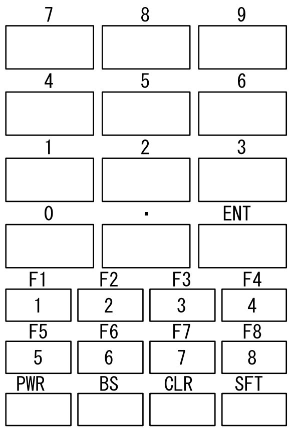
Appendix E
Key Number Assignment on the Keyboard

E1. BHT-3000

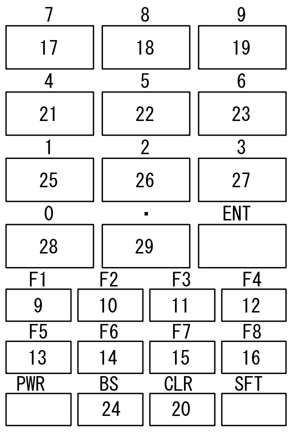
■ Key Number Assignment

The keys on the BHT-3000 keyboard are assigned numbers as shown below.

Non-shift mode



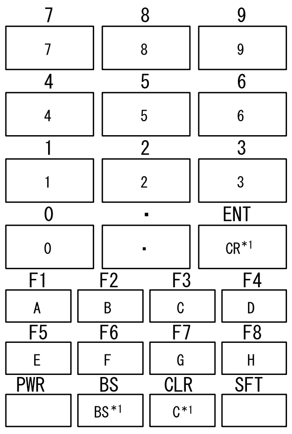
Shift mode



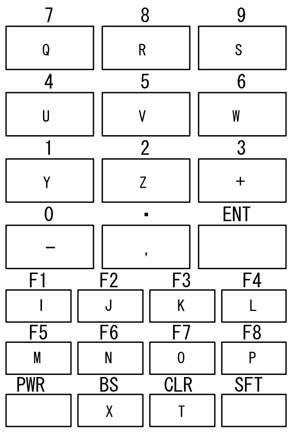
■ Default Data Assignment

The default data assignment is shown below.

Non-shift mode



Shift mode



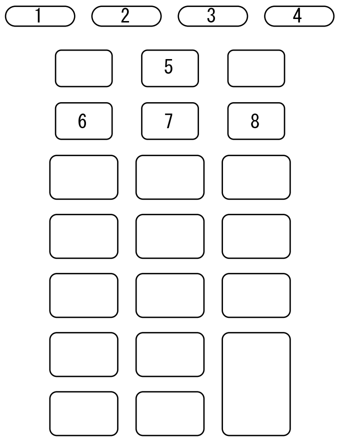
*1 BS, CR, and C are a backspace (08h), carriage return (0Dh), and cancel (18h) code, respectively.

E2. BHT-4000

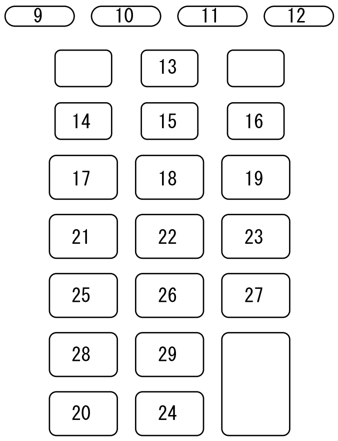
■ Key Number Assignment

The keys on the BHT-4000 keyboard are assigned numbers as shown below.

Non-shift mode



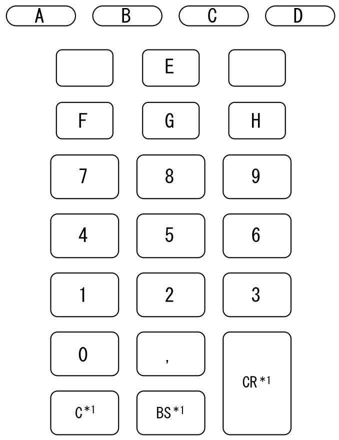
Shift mode



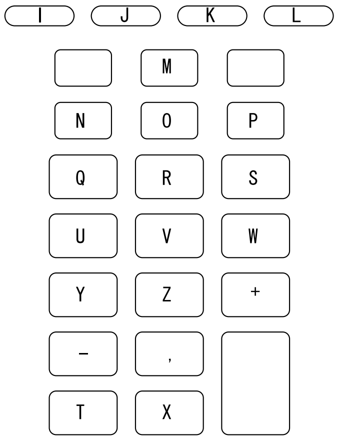
■ Default Data Assignment

The default data assignment is shown below.

Non-shift mode



Shift mode



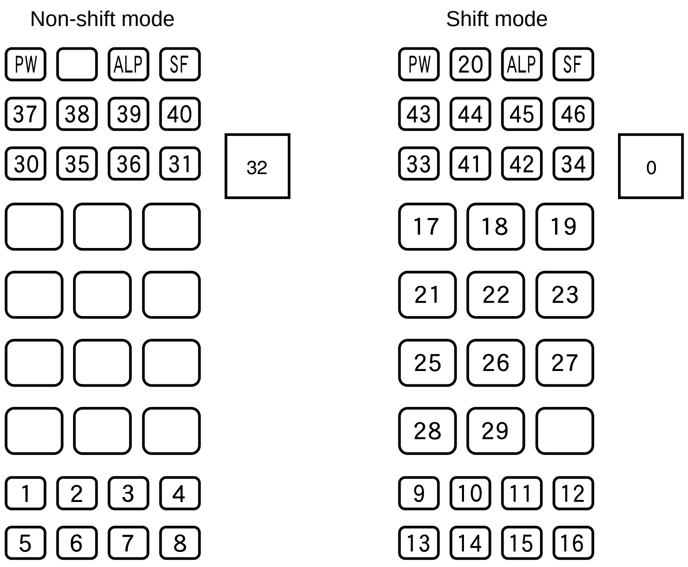
[*1] BS, CR, and C are a backspace (08h), carriage return (0Dh), and cancel (18h) code, respectively.

E3. BHT-5000

[1] 32-key pad

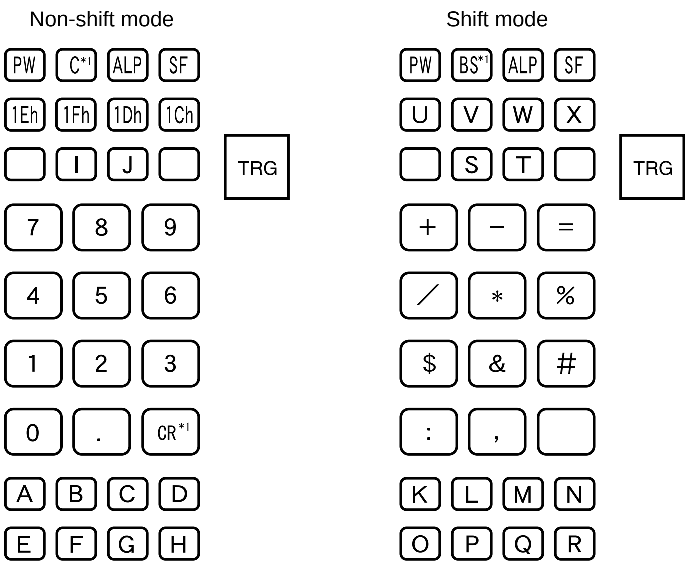
■ Key Number Assignment

The keys on the BHT-5000 keyboard are assigned numbers as shown below.



■ Default Data Assignment

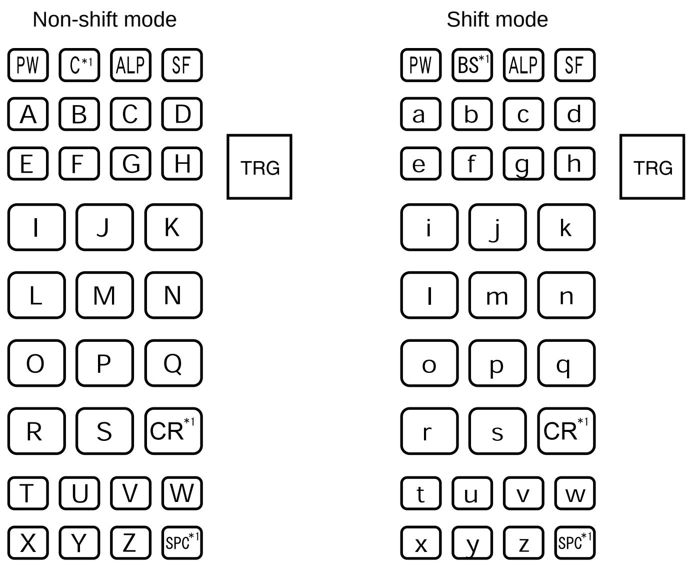
The default data assignment is shown below.



*1 BS, CR, and C are a backspace (08h), carriage return (0Dh), and cancel (18h) code, respectively.

■ Alphabet Letter Assignment

Shown below are the alphabet letter assignments which are available when the alphabet input function is activated.



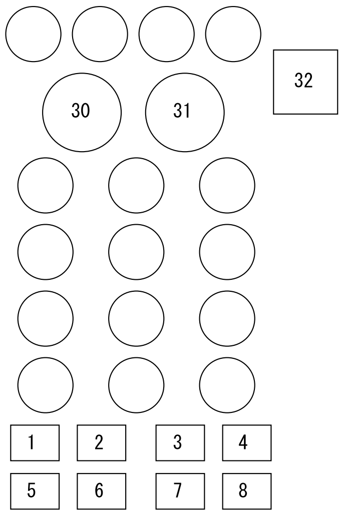
*1 BS, CR, C, and SPC are a backspace (08h), carriage return (0Dh), cancel (18h), and space (20h) code, respectively.

[2] 26-key pad

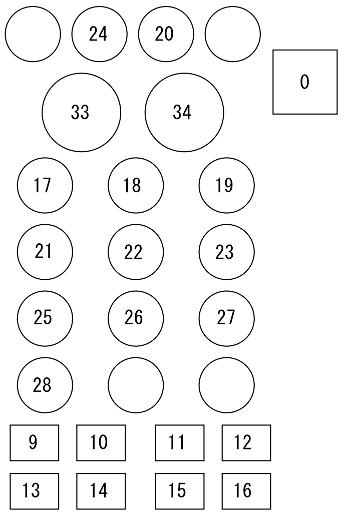
■ Key Number Assignment

The keys on the BHT-5000 keyboard are assigned numbers as shown below.

Non-shift mode



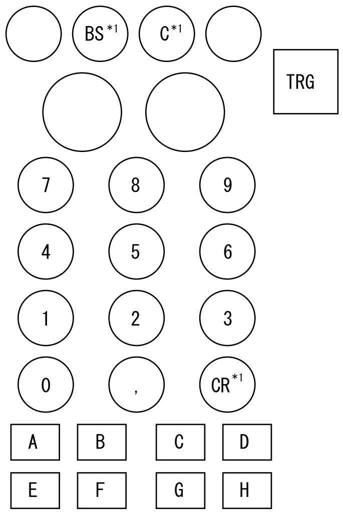
Shift mode



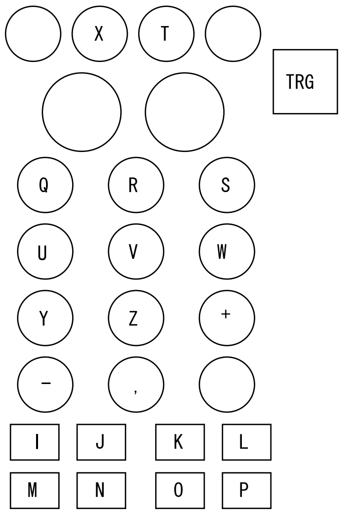
■ Default Data Assignment

The default data assignment is shown below.

Non-shift mode



Shift mode



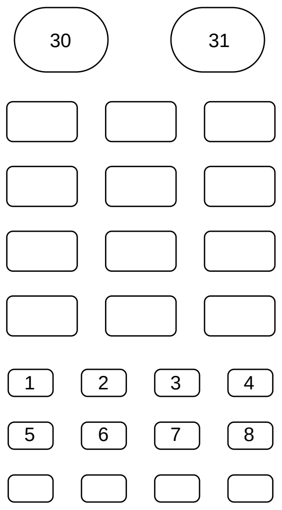
*1 BS, CR, and C are a backspace (08h), carriage return (0Dh), and cancel (18h) code, respectively.

E4. BHT-6000

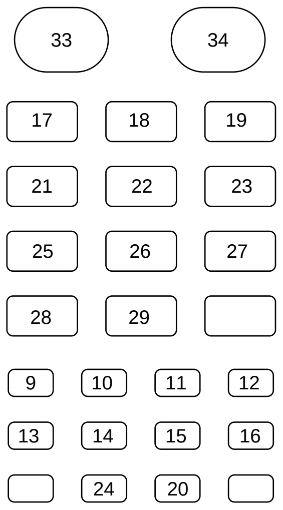
■ Key Number Assignment

The keys on the BHT-6000 keyboard are assigned numbers as shown below.

Non-shift mode



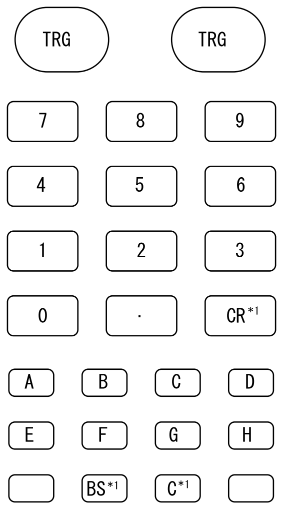
Shift mode



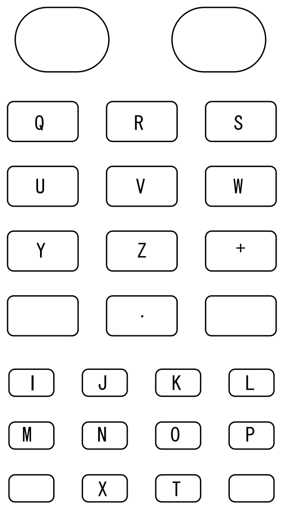
■ Default Data Assignment

The default data assignment is shown below.

Non-shift mode



Shift mode



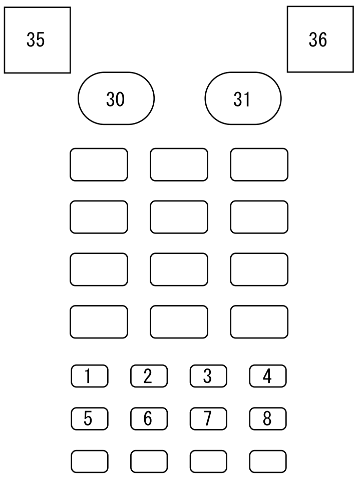
*1 BS, CR, and C are a backspace (08h), carriage return (0Dh), and cancel (18h) code, respectively.

E5. BHT-6500

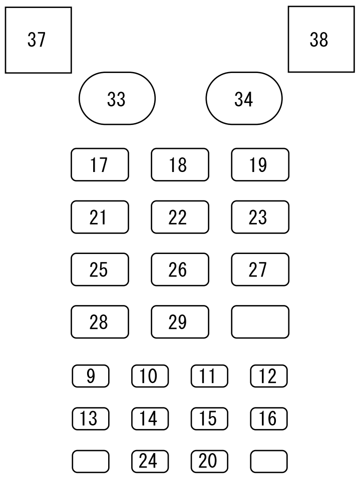
■ Key Number Assignment

The keys on the BHT-6500 keyboard are assigned numbers as shown below.

Non-shift mode



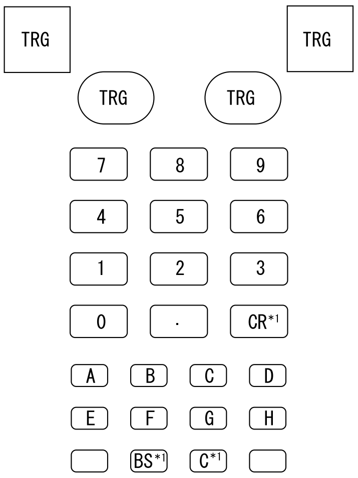
Shift mode



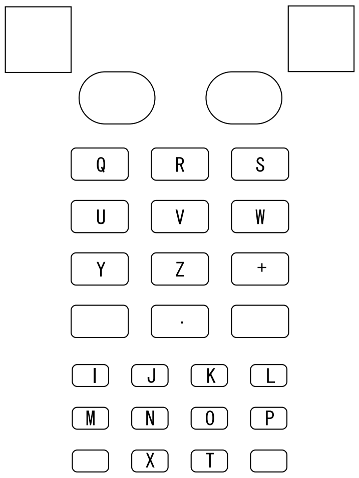
■ Default Data Assignment

The default data assignment is shown below.

Non-shift mode



Shift mode



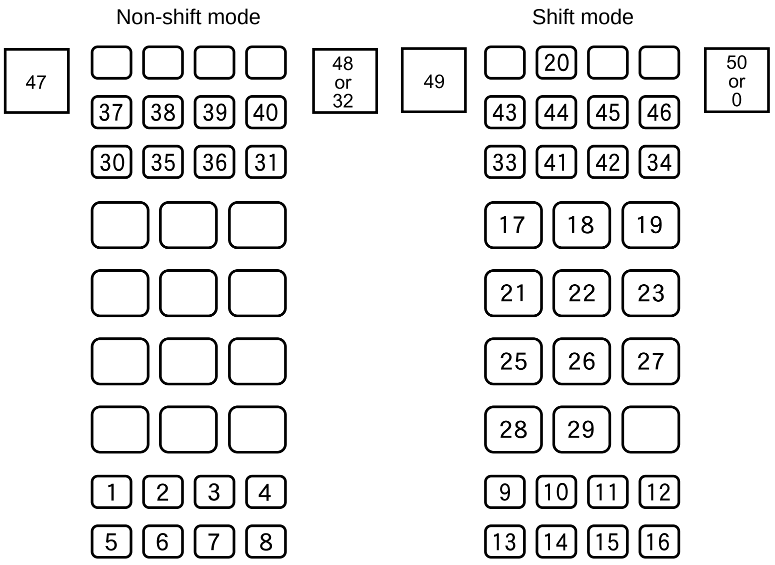
*1 BS, CR, and C are a backspace (08h), carriage return (0Dh), and cancel (18h) code, respectively.

E6. BHT-7000/BHT-7500

[1] 32-key pad

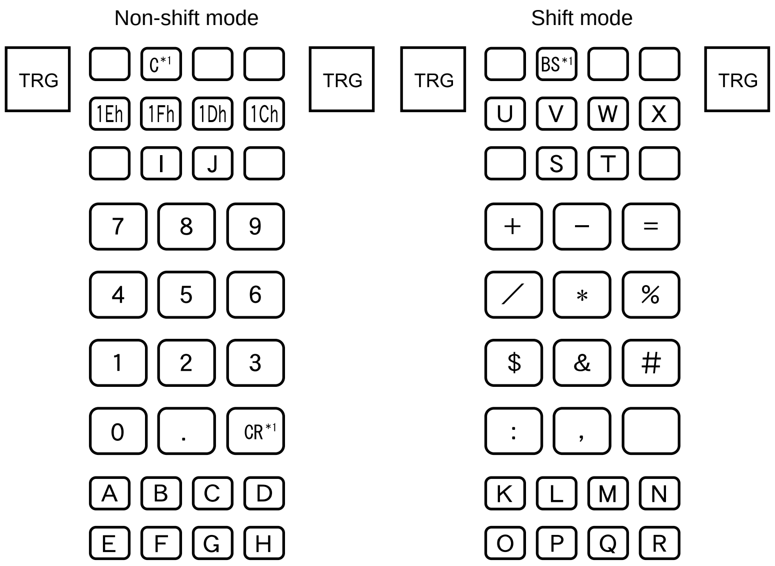
■ Key Number Assignment

The keys on the BHT-7000/BHT-7500 keyboard are assigned numbers as shown below.



■ Default Data Assignment

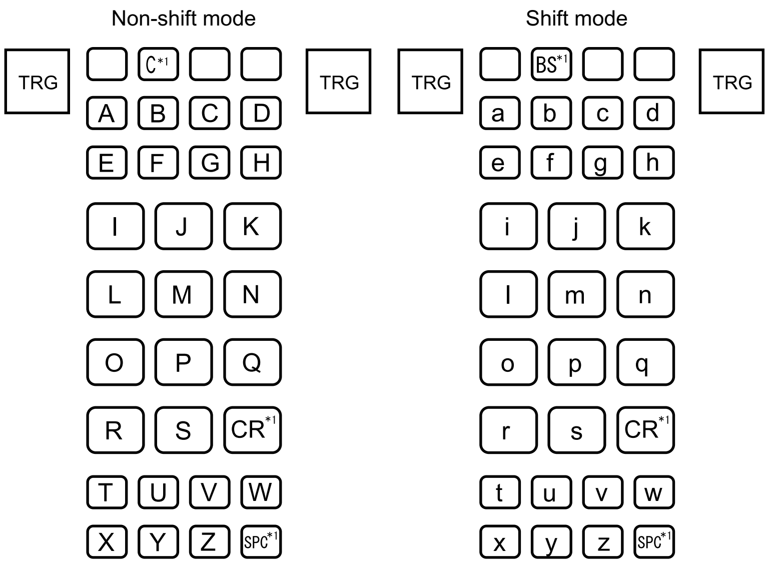
The default data assignment is shown below.



*1 BS, CR, and C are a backspace (08h), carriage return (0Dh), and cancel (18h) code, respectively.

■ Alphabet Letter Assignment

Shown below are the alphabet letter assignments which are available when the alphabet input function is activated.

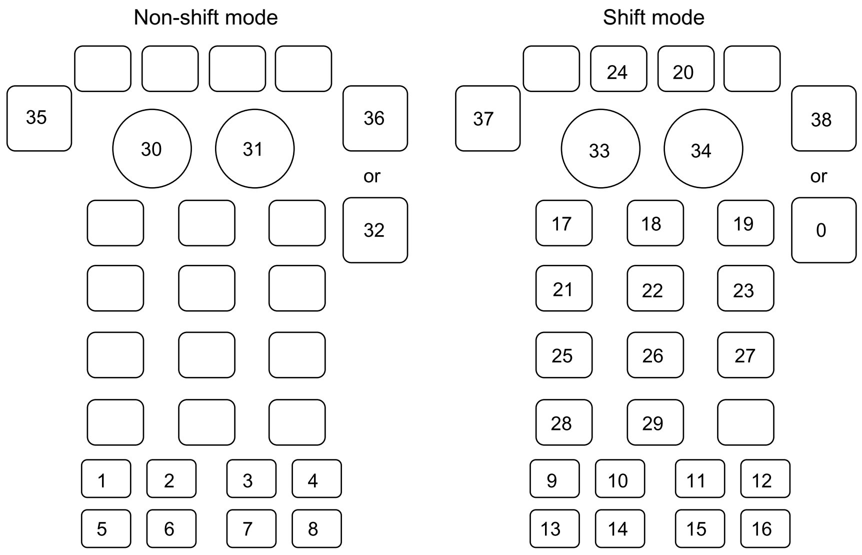


*1 BS, CR, C, and SPC are a backspace (08h), carriage return (0Dh), cancel (18h), and space (20h) code, respectively.

[2] 26-key pad (BHT-7000 only)

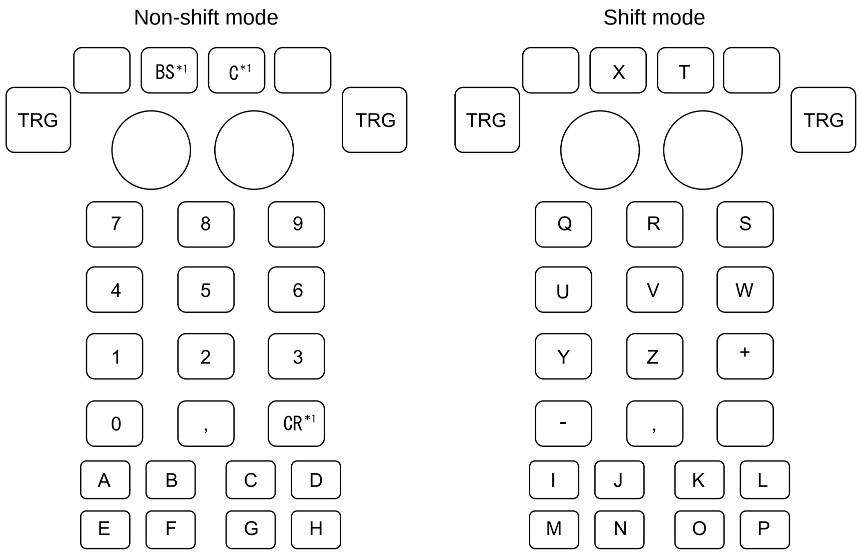
■ Key Number Assignment

The keys on the BHT-7000 keyboard are assigned numbers as shown below.



■ Default Data Assignment

The default data assignment is shown below.

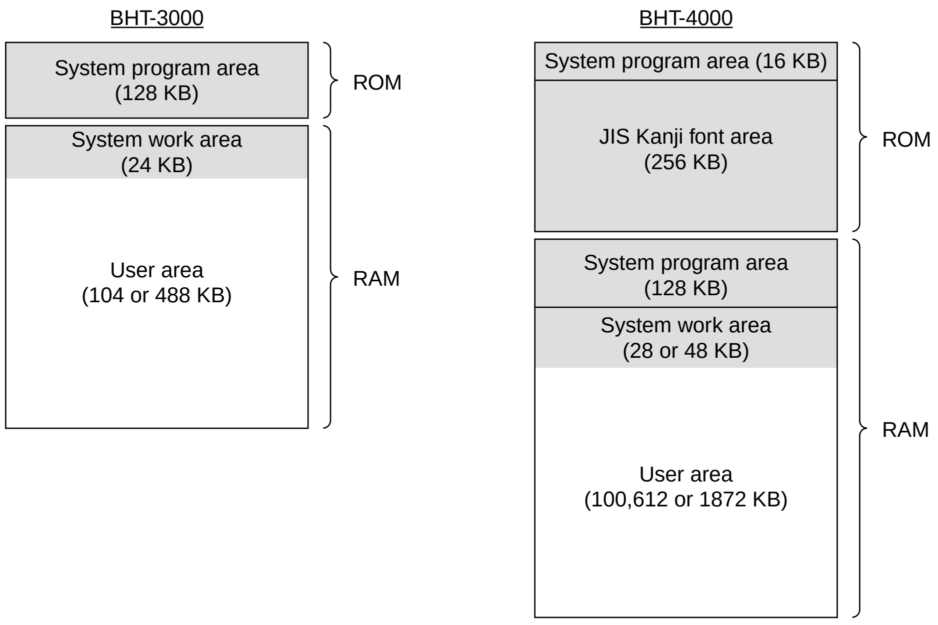


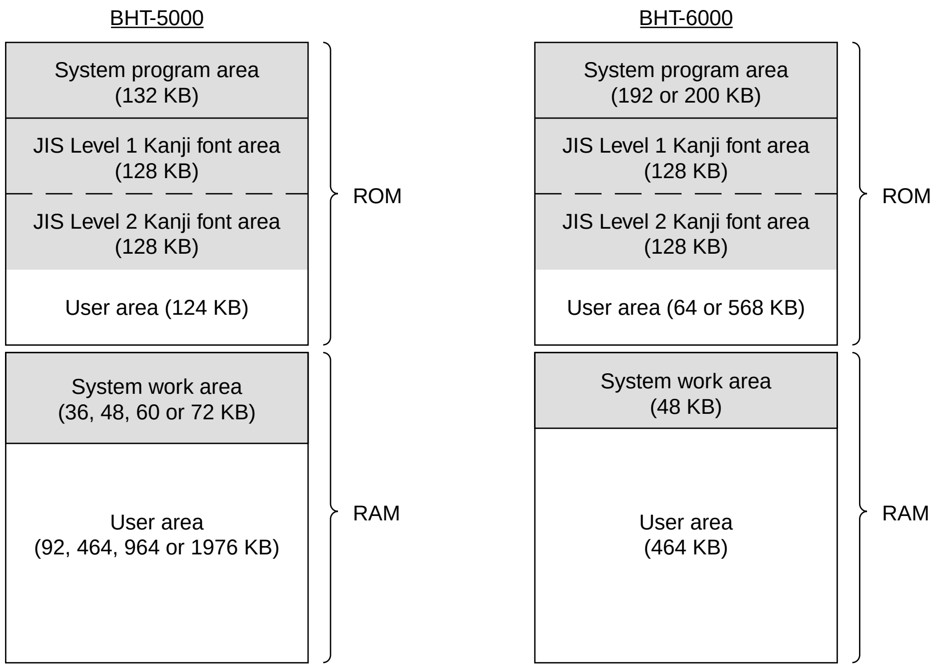
[*1] BS, CR, and C are a backspace (08h), carriage return (0Dh), and cancel (18h) code, respectively.

Appendix F
Memory Area

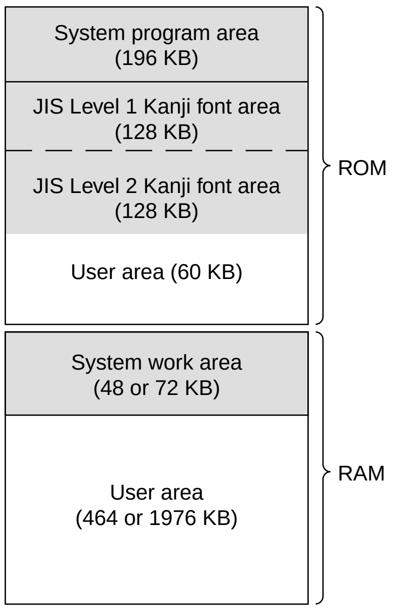
■ Memory Map

The memory maps are shown below.

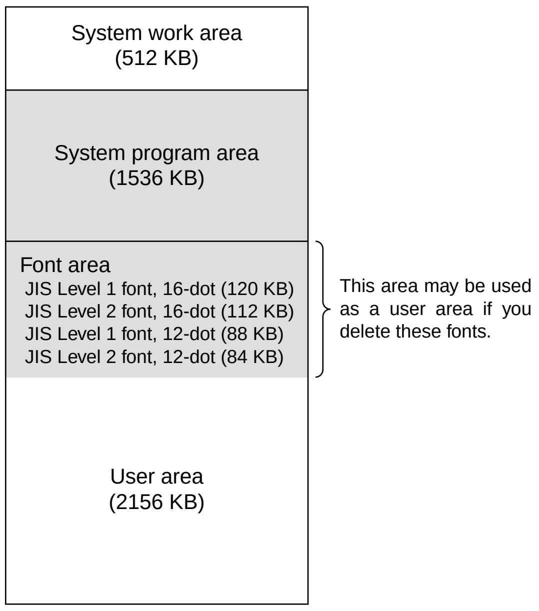

BHT-3000
System program area (128 KB)
ROM
System work area (24 KB)
User area (104 or 488 KB)
RAM
BHT-4000
System program area (16 KB)
JIS Kanji font area (256 KB)
ROM
System program area (128 KB)
System work area (28 or 48 KB)
User area (100,612 or 1872 KB)
RAM


BHT-5000
System program area (132 KB)
JIS Level 1 Kanji font area (128 KB)
JIS Level 2 Kanji font area (128 KB)
User area (124 KB)
ROM
System work area (36, 48, 60 or 72 KB)
User area (92, 464, 964 or 1976 KB)
RAM
BHT-6000
System program area (192 or 200 KB)
JIS Level 1 Kanji font area (128 KB)
JIS Level 2 Kanji font area (128 KB)
User area (64 or 568 KB)
ROM
System work area (48 KB)
User area (464 KB)
RAM

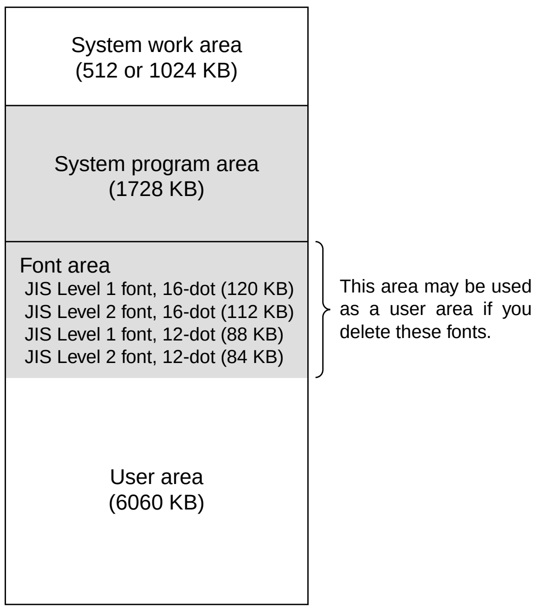
BHT-6500


System program area (196 KB)
JIS Level 1 Kanji font area (128 KB)
JIS Level 2 Kanji font area (128 KB)
User area (60 KB)
ROM
System work area (48 or 72 KB)
User area (464 or 1976 KB)
RAM

BHT-7000


System work area (512 KB)
System program area (1536 KB)
Font area
JIS Level 1 font, 16-dot (120 KB)
JIS Level 2 font, 16-dot (112 KB)
JIS Level 1 font, 12-dot (88 KB)
JIS Level 2 font, 12-dot (84 KB)
This area may be used as a user area if you delete these fonts.
User area (2156 KB)

BHT-7500


System work area (512 or 1024 KB)
System program area (1728 KB)
Font area
JIS Level 1 font, 16-dot (120 KB)
JIS Level 2 font, 16-dot (112 KB)
JIS Level 1 font, 12-dot (88 KB)
JIS Level 2 font, 12-dot (84 KB)
This area may be used as a user area if you delete these fonts.
User area (6060 KB)

The size and area allocation of the memory incorporated in the BHT differ depending upon the models as listed below.

BHT series	Models	User area	User area in drive B (B:)
BHT-3000	BHT-3041	104	
	BHT-3045	488	
BHT-4000	BHT-4082	100	
	BHT-4086	612	
	BHT-4089	1872 *1	
BHT-5000	BHT-5071	92	124 *4
	BHT-5075	464	124 *4
	BHT-5077	964	124 *4
	BHT-5079	1976 *1	124 *4
BHT-6000	BHT-6045	464 *2	64 *4
	BHT-6047	464 *2	568 *4
	BHT-6049	464 *2	1584 *1*4
BHT-6500	BHT-6505	464	60 *4
	BHT-6509	1976 *1	60 *4
BHT-7000	BHT-7064	2156 *3	
BHT-7500	BHT-7508	6060 *3	

*1The cluster size is 8 KB.

*2468 KB in System version 2.00 or newer

*3Plus a maximum of 404 KB if you delete fonts

*4Plus a maximum of 256 KB if you delete fonts

■ Memory Management

The BHT manages the user area of the memory for user programs and data files by a unit of segment called "cluster." The cluster size is usually 4 kilobytes. In some models or drives, the cluster size is 8 kilobytes as listed above.

The maximum allowable size for a single user program is 64 kilobytes excluding register variables.

■ Battery Backup of Memory

The BHT-3000 backs up user programs and data files stored in the memory with alkaline manganese batteries. The BHT-4000 backs up them with a rechargeable battery cartridge. The BHT-5000/BHT-7000/BHT-7500 backs up them with a rechargeable battery cartridge or dry battery cartridge. The BHT-6000/BHT-6500 backs up them with dry batteries or rechargeable battery cartridge. Therefore, those data will not be lost if the BHT power is turned off.

■ Memory Space Available for Variables

Listed below are the maximum memory spaces available for work, common, and register variables.

Variables	Max. memory space
Work and common variable area	6 KB*
Register variable area	64 KB

* 32 KB in the BHT-7000/BHT-7500

Each variable occupies the memory space as listed below.

Variables	Max. memory space
An integer variable	2 bytes
A real variable	6 bytes
A string variable	2 to 256 bytes (Including a single character count byte)

An array variable occupies the memory space by (number of bytes per array element x number of array elements).

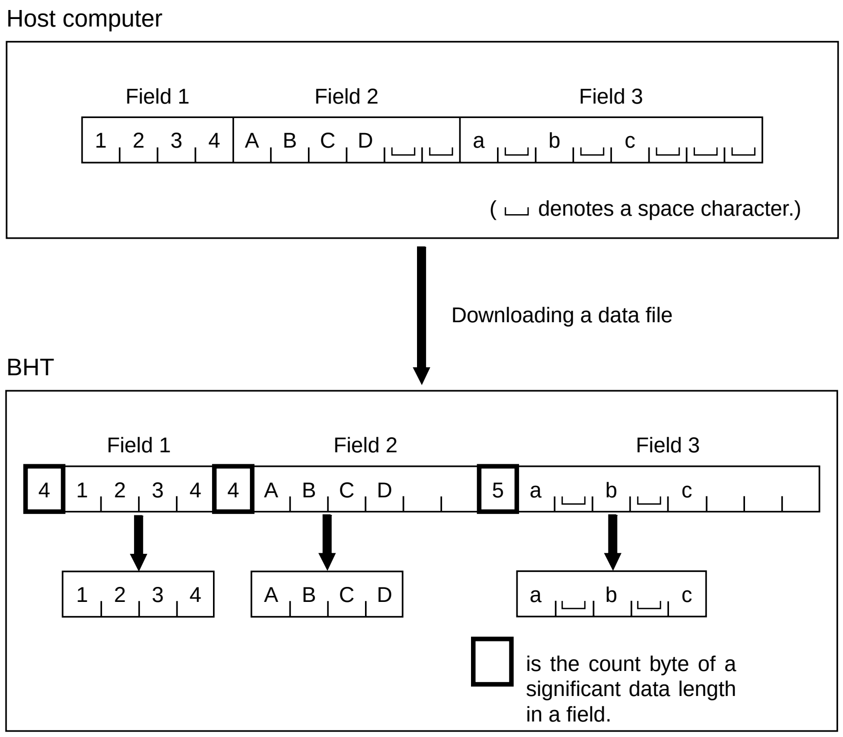
Appendix G Handling Space Characters in Downloading

■ Space characters used as padding characters

A data file can be downloaded with System Mode or an `XFILE` statement according to the communications protocol which is designed to eliminate space characters padded in the tail of each data field. That is, such space characters in a data file will not be handled as data in the BHT-3000/BHT-4000 since the BHT-3000/BHT-4000 has no feature for regenerating those eliminated ones automatically.

The BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500 has a new feature which can handle space characters placed in the tail of a data field as data.

The figure below shows the process in which the space characters used as padding characters are eliminated. (Note that space characters between a and b and between b and c in field 3 are not padding characters.)

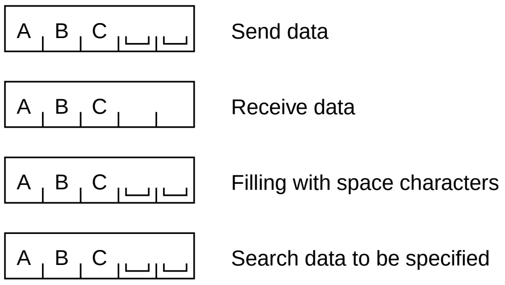


■ To handle space characters as data

To handle space characters in the tail of a data field as data (not as padding characters), you must take special considerations in programming.

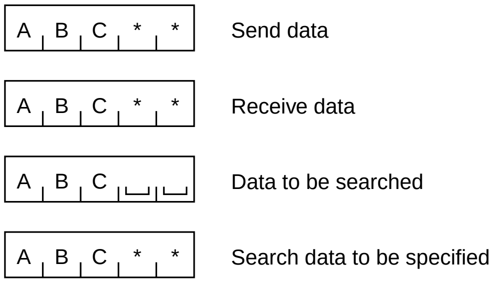
If you want to search for a field data containing space characters in its tail by using a `SEARCH` function, for instance, use any of the following methods:

[Example 1] After downloading a data file, fill the unused spaces in each field with space characters and then search for the target field data.



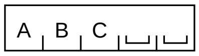
(␣ denotes a space character.)

[Example 2] Before downloading a data file, substitute any of the characters which will not be used as effective data, e.g., an asterisk (*), for the space characters in the host computer.

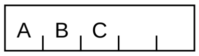


(␣ denotes a space character.)

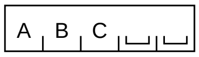
[Example 3] When specifying a field data to be searched, do not include space characters in the tail of the data field.



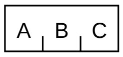
Send data



Receive data



Data to be searched



Search data to be specified

(␣ denotes a space character.)

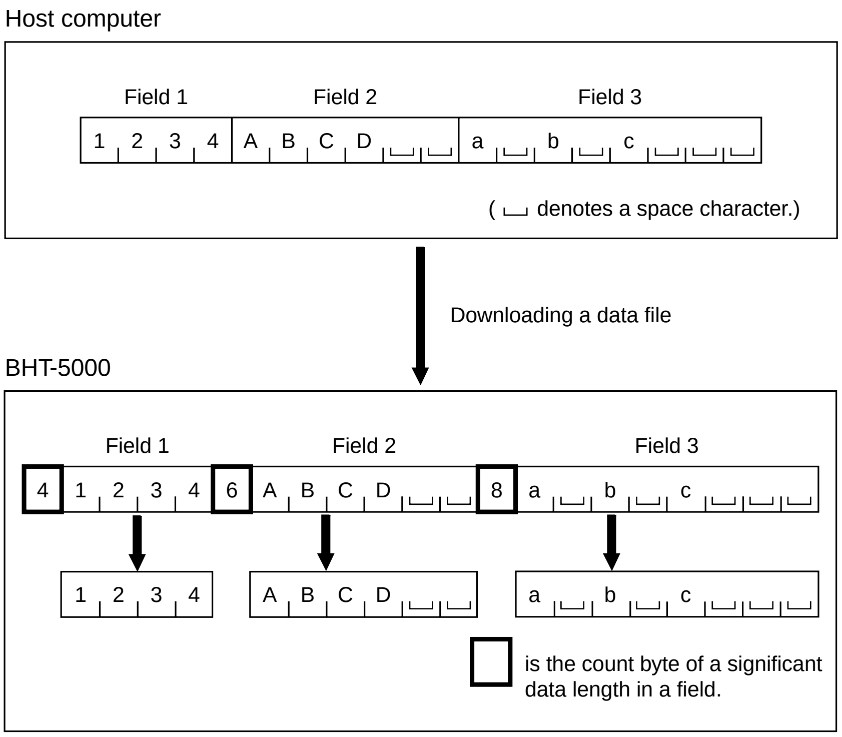
■ To make the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500 handle space characters as data

You can specify the handling of space characters in the tail of a data field with System Mode or an `XFILE` statement.

System Mode:	To handle space characters as data, select "Data" on the field space setting screen on the communications parameter setting menu called up from the SET SYSTEM menu.
`XFILE` statement:	To handle space characters as data, specify T to "*protocolspec*" in the `XFILE` statement.

```
XFILE "d2.dat","T"
```

The figure below shows the process in which the space characters in the tail of a data field are handled as data in the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500.



Appendix H
Programming Notes

[1] Flash ROM

■ BHT-5000/BHT-6000/BHT-6500

You can store user program files and data files in the flash ROM as well as in the RAM. The following tips help you use the flash ROM correctly.

(1) Memory areas required for user programs

If you store a user program into the flash ROM, the area for its register variables is also reserved in the flash ROM. When starting the user program for the first time, the Interpreter copies the register variables stored in the flash ROM into the RAM (so that both the flash ROM and RAM store the register variables). The user program uses the register variables stored in the RAM.

That is, a user program even stored in the flash ROM requires the RAM area for storing its register variables. If the RAM has no sufficient area for storing the register variables, a run-time error will occur.

When uploading a program file stored in the flash ROM, the BHT-5000/BHT-6000/BHT-6500 combines the program (except for the register variables in the flash ROM) with the register variables stored in the RAM.

(2) Retained contents of the flash ROM

Files stored in the RAM are backed up by the built-in rechargeable lithium battery. It means that those files may be damaged if the BHT-5000/BHT-6000/BHT-6500 is left unused for a long time so that the battery voltage drops below the specified level.

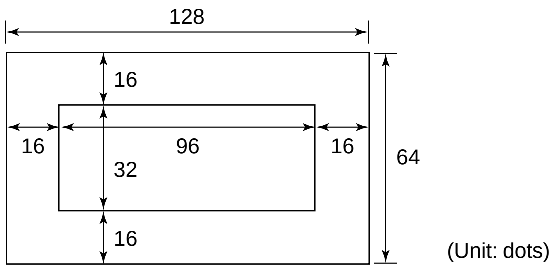
Unlike files stored in the RAM, files stored in the flash ROM are retained independently of the voltage level of the lithium battery. Once data is written onto the flash ROM, it will be retained until you delete it.

[2] BHT-2000 compatible mode

■ BHT-5000

You can run user programs written for the BHT-2000 on the BHT-5000 without any program modification if you select the BHT-2000 compatible mode on the OTHERS menu of the SET SYSTEM menu in System Mode.

When those user program are running, they appear only in the middle section of the LCD as shown below. This is because the BHT-5000 is larger than the BHT-2000 in the numbers of columns and lines.



The following items are not compatible in the BHT-2000 compatible mode:

(1) Frequencies of the beeper when 0, 1, or 2 is set to the `frequency` option in the BEEP statement

(2) Auto-repeat of keys

		BHT-2000	BHT-2000 compatible mode
Frequency	0	1046 Hz	1015 Hz
	1	2092 Hz	2142 Hz
	2	3922 Hz	4200 Hz
Keys		Auto-repeat	Not auto-repeat

NOTE Some user programs written for the BHT-2000 may not work correctly in the BHT-2000 compatible mode.

Before the practical use of user programs written for the BHT-2000 in this mode, check the program operation sufficiently.

[3] Program file named APLINT.PD3

■ BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500

If a program file named APLINT.PD3 is stored in the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, the System Mode initiation sequence (by pressing the PW key with the SF and 1 keys held down) will not start System Mode but execute that user program.

Making a program file named APLINT.PD3 allows you to:

- enter an ID number at the start of System Mode and
- set the condensed System Mode which is used for maintenance of user programs.

To terminate the APLINT.PD3 file, you use the `END` or `POWER OFF` statement. When terminating the file with the `END` statement, you may start System Mode by setting the port 6000h as listed below.

Port No.	Bit assignment	Controls the following:
6000h	0	0: Does not start System Mode (default) 1: Starts System Mode

Appendix I
Program Samples

Writing the function for receiving both bar code entry and key entry

Feature:	This function receives earlier one of either bar code entry or key entry. If bar code reading is completed, the function returns the scanned bar code data; if key entry comes first, the function inhibits bar code reading and echoes back the key entry data, then returns the key entry data when the ENT key is pressed. If pressing the Backspace key or Clear key makes the input string empty, then the function becomes ready to receive the subsequent bar code entry or key entry.
Returned value:	The function returns bar code data or key entry data which has come in until the ENT key is pressed, as a string.
Arguments:	`f.no%` Specifies the file number which opens the bar code device file. (Invariant allowed) `bar$` Specifies bar code reading. (Invariant allowed) Ex. "M:10-20" `max%` Specifies the maximum length of a returned string `esc$` If a key(s) contained in this string is entered, the function returns the key entry only.
Work:	`.kb$` and `.rt$`

If you use an invariant for `f.no%` or `bar$`, it is not necessary to pass the value as an argument.

The `bar$` can pass a single type of bar code. If two or more types are required, directly describe necessary invariants.

```
def fnbarkey$(f. no%, bar$, max%, esc$)
   while 1
      open "BAR:" as #f. no% code bar$
      wait 0, 3                ' Wait for completion of bar code reading or key press.
      if loc(#f. no%) then
         beep                  ' Beep when bar code reading is completed.
         fnbarkey$ = input$(max%, #f. no%)
                               ' For displaying:
                               ' rt$ = input$(max%, #f. no%) : print .rt$;
                               ' fnbarkey$ = .rt$
         close #f. no%
         exit def
      else
         close #f. no%                      ' Receive only key entry.
         .rt$ = ""
         .kb$ = input$(1)
         while .kb$<>""
            if instr(esc$, .kb$) then      ' Key designated in esc$?
```

```
          fnbarkey$ = .kb$          ' Then, return the character.
          exit def
        endif
        select .kb$
        case chr$(13)
          fnbarkey$ = .rt$
          exit def
        case chr$(8)                ' BS key.
          if len(.rt$) then
            print chr$(8);          ' Erase one character.
            .rt$ = left$(.rt$, len(.rt$)-1)
          endif
        case chr$(24)               ' Clear key.
          while len(.rt$)           ' Erase all characters entered.
            print chr$(8);
            .rt$ = left$(.rt$, len(.rt$)-1)
          wend
        case else
          if len(.rt$)<max% then
                                    ' Check if only numeric data should be
                                    ' received.
            print .kb$;             ' Echo back.
            .rt$ = .rt$ + .kb$
          else
            beep                    ' Exceeded number of characters error.
          endif
        end select
        if .rt$="" then             ' If input string is empty, go back to the
                                    ' initial state.
          .kb$ = ""
        else
          .kb$ = input$(1)          ' Subsequent key entry.
        end if
      wend
    endif
  wend
end def
```

Testing the written function

```
while 1                                     'Infinite loop
   a$ = fnbarkey$ (1, "A", 15, "DL")        'F4 and SFT/F4 as escape characters.
   print
   if a$<>"D" and a$<>"L" then
      print "Data="; a$
   else
      print "ESC(";a$;") key push"
   endif
wend
end
```

Appendix J Quick Reference for Statements and Functions

Controlling program flow

Statements

`CALL`	Calls an FN3 or SUB function.
`CHAIN`	Transfers control to another program.
`END`	Terminates program execution.
`FOR...NEXT`	Defines a loop containing statements to be executed a specified number of times.
`GOSUB`	Branches to a subroutine.
`GOTO`	Branches to a specified label.
`IF...THEN...ELSE...END IF`	Conditionally executes specified statement blocks depending upon the evaluation of a conditional expression.
`ON...GOSUB`	Branches to one of specified labels according to the value of an expression.
`ON...GOTO`	Branches to one of specified labels according to the value of an expression.
`RETURN`	Returns control from a subroutine or an event-handling routine (for keystroke interrupt).
`SELECT...CASE...END SELECT`	Conditionally executes one of statement blocks depending upon the value of an expression.
`WHILE...WEND`	Continues to execute a statement block as long as the conditional expression is true.

Handling errors

Statements

`ON ERROR GOTO`	Enables error trapping.
`RESUME`	Causes program execution to resume at a specified location after control is transferred to an error-handling routine.

Functions

`ERL`	Returns the current statement location of the program where a run-time error occurred.
`ERR`	Returns the error code of the most recent run-time error.

Defining and allocating variables

Statements

`COMMON`	Declares common variables for sharing between user programs.
`CONST`	Defines symbolic constants to be replaced with labels.
`DATA`	Stores numeric and string literals for `READ` statements.
`DECLARE`	Declares user-created function FUNCTION or SUB externally defined.
`DEFREG`	Defines register variables.
`DIM`	Declares and dimensions arrays; also declares the string length for a string variable.
`ERASE`	Erases array variables.
`GLOBAL`	Declares one or more work variables or register variables defined in a file, to be global.
`LET`	Assigns a value to a given variable.
`PRIVATE`	Declares one or more work variables or register variables defined in a file, to be private.
`READ`	Reads data defined by `DATA` statement(s) and assigns them to variables.
`RESTORE`	Specifies a `DATA` statement location where the READ statement should start reading data.

Controlling the LCD screen

Statements

`APLOAD`	Loads a user-defined font in the single-byte ANK mode.
`CLS`	Clears the LCD screen.
`CURSOR`	Turns the cursor on or off.
`KEY`	Assigns a string or a control code to a function key; also defines a function key as the LCD backlight function on/off key. This statement also defines a magic key as the trigger switch, shift key, or battery voltage display key.
`KPLOAD`	Loads a user-defined Kanji font in the two-byte Kanji mode. This statement also loads a user-defined cursor for the BHT-7000/BHT-7500.
`LOCATE`	Moves the cursor to a specified position and changes the cursor shape.
`PRINT`	Displays data on the LCD screen.
`PRINT USING`	Displays data on the LCD screen under formatting control.
`SCREEN`	Sets the screen mode and the character attribute.

Functions

`COUNTRY$`	Sets a national character set or returns a current country code.
`CSRLIN`	Returns the current row number of the cursor.
`POS`	Returns the current column number of the cursor.

Controlling the keyboard input

Statements

`INPUT`	Reads input from the keyboard into a variable.
`KEY`	Assigns a string or a control code to a function key; also defines a function key as the LCD backlight function on/off key. This statement also defines a magic key as the trigger switch, shift key, or battery voltage display key.
`KEY ON`	Enables keystroke trapping for a specified function key.
`KEY OFF`	Disables keystroke trapping for a specified function key.
`LINE INPUT`	Reads input from the keyboard into a string variable.
`ON KEY...GOSUB`	Specifies an event-handling routine for keystroke interrupt.

Functions

`INKEY$`	Returns a character read from the keyboard.
`INPUT$`	Returns a specified number of characters read from the keyboard or from a device file.

Beeping

Statements

`BEEP`	Drives the beeper or vibrator. (The vibrator is provided in the BHT-6500/BHT-7000/BHT-7500.)

Manipulating the system date, the current time, or the timers

Functions

`DATE$`	Returns the current system date or sets a specified system date.
`TIME$`	Returns the current system time or wakeup time, or sets a specified system time or wakeup time.
`TIMEA`	Returns the current value of timer A or sets timer A.
`TIMEB`	Returns the current value of timer B or sets timer B.
`TIMEC`	Returns the current value of timer C or sets timer C.

Communicating with I/Os

Statements	
`OUT`	Sends a data byte to an output port.
`POWER`	Controls the automatic power-off facility.
`WAIT`	Pauses program execution until a designated input port presents a given bit pattern.
Functions	
`FRE`	Returns the number of bytes available in a specified area of the memory.
`INP`	Returns a byte read from a specified input port.

Communicating with the barcode device

Statements	
`CLOSE`	Closes file(s).
`INPUT#`	Reads data from a device I/O file into specified variables.
`OPEN "BAR:"`	Opens the bar code device file. In the BHT-5000/BHT-6000/BHT-6500/BHT-7000/BHT-7500, this statement also activates or deactivates the reading confirmation LED and the beeper (vibrator) individually. (Vibrator control valid only in the BHT-6500/BHT-7000/BHT-7500)
Functions	
`CHKDGT$`	Returns a check digit of bar code data.
`EOF`	Tests whether the end of a device I/O file has been reached.
`INPUT$`	Returns a specified number of characters read from the keyboard or from a device file.
`LOC`	Returns the current position within a specified file.
`MARK$`	Returns a bar code type and the number of digits of the bar code.

Manipulating data files and user program files

Statements	
`CLFILE`	Erases the data stored in a data file.
`CLOSE`	Closes file(s).
`FIELD`	Allocates string variables as field variables.
`GET`	Reads a record from a data file.
`KILL`	Deletes a specified file from the memory.
`OPEN`	Opens a file for I/O activities.
`PUT`	Writes a record from a field variable to a data file.
Functions	
`LOC`	Returns the current position within a specified file.
`LOF`	Returns the length of a specified file.
`SEARCH`	Searches a specified data file for specified data, and then returns the record number where the search data is found.

Communicating with communications devices

Statements	
`CLOSE`	Closes file(s).
`INPUT#`	Reads data from a device I/O file into specified variables.
`LINE INPUT#`	Reads data from a device I/O file into a string variable.
`OPEN "COM:"`	Opens a communications device file.
`PRINT#`	Outputs data to a communications device file.
`XFILE`	Transmits a designated file according to the specified communications protocol.

Functions

`BCC$`	Returns a block check character (BCC) of a data block.
`EOF`	Tests whether the end of a device I/O file has been reached.
`ETX$`	Modifies the value of a terminator (ETX) for the BHT-protocol; also returns the current value of a terminator.
`INPUT$`	Returns a specified number of characters read from the keyboard or from a device file.
`LOC`	Returns the current position within a specified file.
`LOF`	Returns the length of a specified file.
`SOH$`	Modifies the value of a header (SOH) for the BHT-protocol; also returns the current value of a header.
`STX$`	Modifies the value of a header (STX) for the BHT-protocol; also returns the current value of a header.

Commenting a program

Statements

`REM`	Declares the rest of a program line to be remarks or comments.

Manipulating numeric data

Functions

`ABS`	Returns the absolute value of a numeric expression.
`INT`	Returns the largest whole number less than or equal to the value of a given numeric expression.

Manipulating string data

Functions	
`ASC`	Returns the ASCII code value of a given character.
`CHR$`	Returns the character corresponding to a given ASCII code.
`HEX$`	Converts a decimal number into the equivalent hexadecimal string.
`INSTR`	Searches a specified target string for a specified search string, and then returns the position where the search string is found.
`LEFT$`	Returns the specified number of leftmost characters from a given string expression.
`LEN`	Returns the length (number of bytes) of a given string.
`MID$`	Returns a portion of a given string expression from anywhere in the string.
`RIGHT$`	Returns the specified number of rightmost characters from a given string expression.
`STR$`	Converts a numeric expression into a string.
`VAL`	Converts a string into a numeric value.

Defining user-created functions

Statements	
`DEF FN`	Names and defines a user-created function.
`DEF FN…END DEF`	Names and defines a user-created function.
`FUNCTION…END FUNCTION`	Names and defines user-created function FUNCTION.
`SUB…END SUB`	Names and defines user-created function SUB.

Specifying included files

Statements	
`$INCLUDE`	Specifies an included file.
`REM $INCLUDE`	Specifies an included file.

Appendix K
Unsupported Statements and Functions

BHT-BASIC does not support the following MS-BASIC statements and functions:

- For handling sequential data files

CVD	MKD$	PRINT# USING
CVI	MKI$	RSET
CVS	MKS$	WRITE#
LSET	PRINT#	

- For RS-232C interface operation

```
PRINT# USING
WRITE#
```

- For interrupt handling

COM OFF	ON STOP GOSUB
COM ON	STOP OFF
COM STOP	STOP ON
ON STCOM GOSUB	

- For graphics and color control

CIRCLE	DRAW	WIDTH
COLOR	LINE	WINDOW
CONSOLE	POINT	
CSRLIN	PSET	

- For I/O control

DEFUSR	POKE
PEEK	VARPTR

- For mathematical functions and trigonometric functions

ATN	LOG	SQR
COS	SCNG	TAN
EXP	SIN	

- For others

CDBL	FIX	SGN
CINT	IF GOTO	STRING$
CLEAR	LPOS	SWAP
COPY	OCT$	TAB
DEF DBL	OPTION BASE	WRITE
DEF SNG	RANDOMIZE	
DEFINT	RND	

Index

BHT-BASIC

Programmer's Manual

First Edition, May 1993

Fifth Edition, October 2000

DENSO CORPORATION

Industrial Systems Product Division