DENSO

BHT-BASIC

Programmer's Manual

(BHT- 8000 series)

Specifications are subject to change without prior notice.

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-8000.

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 16 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 concate nated 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 Functions

Describes low battery warning, the prohibited simultaneous operation of the beeper / illumination LED, the wakeup, and remote wakeup.

Chapter 13. Backlight Function

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

Chapter 17. TCP/IP (BHTs with Bluetooth communications device)

Surveys the socket application program interface (API) and FTP client. This chapter also describes the two function libraries--SOCKET.FN3 and FTP.FN3, which provide BHT-BASIC programs with access to a subset of the TCP/IP family of protocols.

Chapter 18. Bluetooth (BHTs with Bluetooth communications device)

Describes the Bluetooth communication system and communications programming.

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`

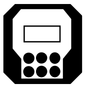
7. Programs make no distinction between uppercase and lowercase letters, except for character string data.

   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.

BHT and CU

This manual expresses BHT-8000 series as "BHT."

The CU-8000 series is expressed as "CU."

■Abbreviations

ANK	Alpha-Numeric 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	Communication 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

BHT-8000 Series User's Manuals
Transfer Utility Guide
Ir-Transfer Utility C Guide
Ir-Transfer Utility E Guide

■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.

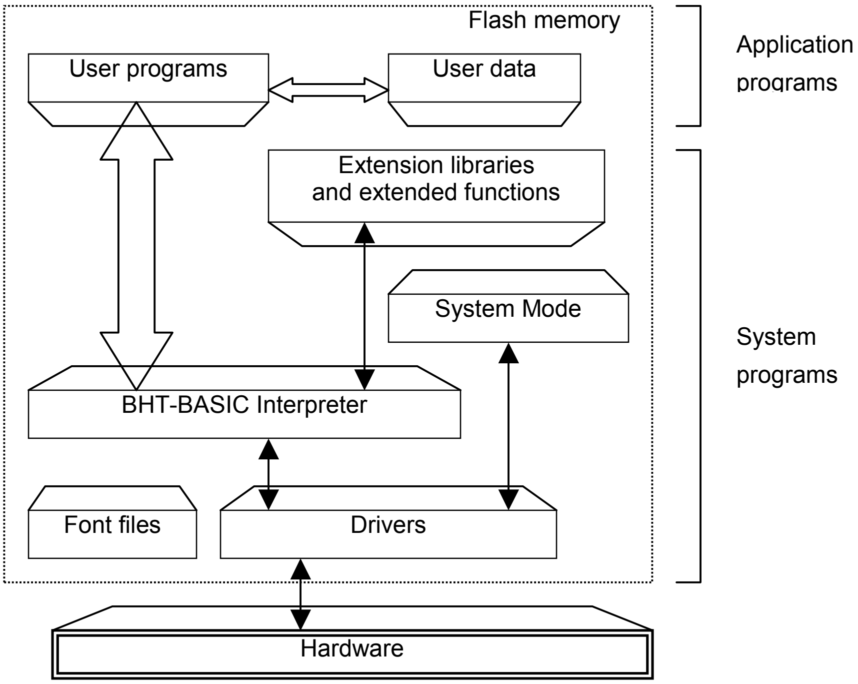
Chapter 1
Software Overview for the BHT

CONTENTS

1.1 Software Overview

1.1.1 Software Structure of the BHT

The structure of software for the BHT is shown below.



The BHT has a flash memory and RAM. All of the system programs, user programs, extension libraries, and extended functions are stored in the flash memory. 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.

System Mode

Sets up the execution environment for user programs.

Extension Library

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

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

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

Use extension libraries suited for BHT-8000.

□Application Programs

User Programs

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

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/NT/2000/XP

You may develop programs with BHT-BASIC on those computers operating on Windows95/98/NT/2000/XP.

■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 pro-grams 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-defined function FUNCTION or SUB externally defined.
FUNCTION...END FUNCTION	Names and defines user-defined 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-defined function SUB.

BHT-BASIC 3.5 provides the constants definition file "BHTDEF.INC." Reading the "BHT-DEF. INC" as an included file allows you to use constant names defined in that file.

Example
```
'$INCLUDE:'BHTDEF.INC'
      OUT .pnLEDCtrl,.pvLEDGrn  'Turn LED (green) ON
```

■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.

■Scoping variables to be local or global (with PRIVATE or GLOBAL statement)

In BHT-BASIC 3.5, work variables and register variables may have "scope" to restrict the access to them.

With the PRIVATE statement, you may declare a variable to be local. A local variable can only be accessed by any routine in a file where it is defined. With the GLOBAL statement, you may declare a variable to be global. A global variable can be accessed by any routine in a program.

However, a variable used inside the FUNCTION or SUB function without declaration is available only within a function where it is defined.

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

For details about the scope of variables, refer to Chapter 5, Section 5.5.

■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/NT/2000/XP.

■BHT (Bar code handy terminal)

- BHT-8000 series

■CU (Optical communications unit)

For IrDA communication, the following CU is required. Note that no CU is required if the BHT is directly connected with the host computer via the direct-connect interface.

- CU-8000 (Option. 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 User's Manual.)

2.1.2 Required Software

• OS	Windows95/98/NT/2000/XP	
• 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 com-pressed 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 END IF, 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]
```

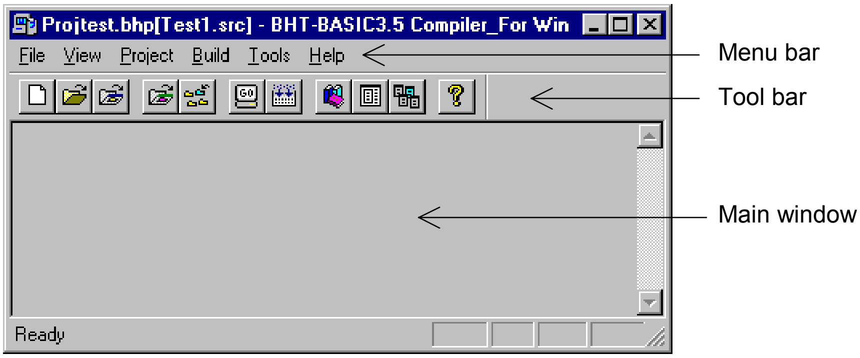
NOTE

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.



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 Open Close Open Project Close Project Exit	 (Yellow)	Creates a new project. Opens an existing file. Closes the active file. Opens an existing project. Closes the active project. Quits the BHT-BASIC 3.5 Compiler.
View	Toolbar Status Bar Clear Screen		Shows or hides the toolbar. Shows or hides the status bar. Clears the screen.
Project	Select File Add File	 (Red)	Selects or deletes a file in the active project. Adds one or more files to the active project.
Build	Compile Build		Compiles one or more active files (or active project) to produce an object file(s). Compiles one or more active files (or active project) and then links them to produce a user program.
Tools	Options Run Editor Set Editor		Sets compiling options and linking options. Runs the 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 pro-grams (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.

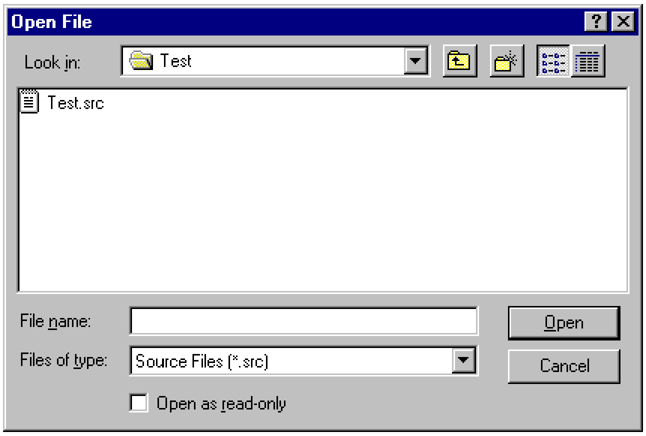
[1] Select a source file

(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

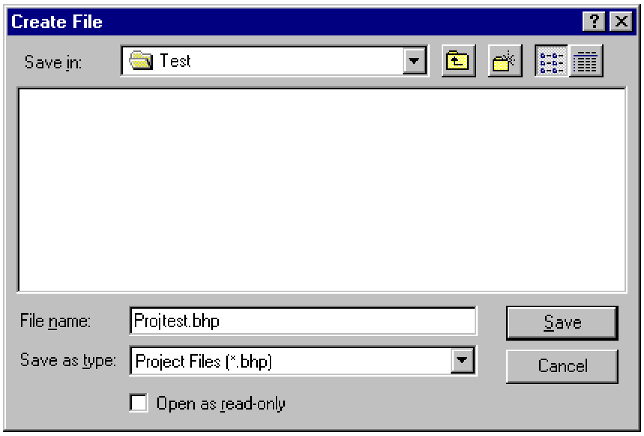
To build a library out of a single source file or to build a user program or library out of multiple source files, you need to create a project file (described in [1] later) or select an existing project file (in [2]).

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.



(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 instructions 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.

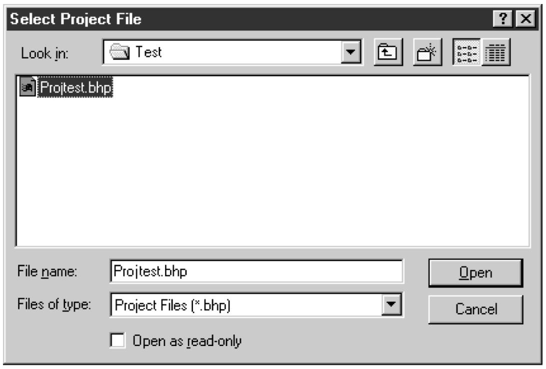
<u>Selecting in the Select Project File dialog box</u>

(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.



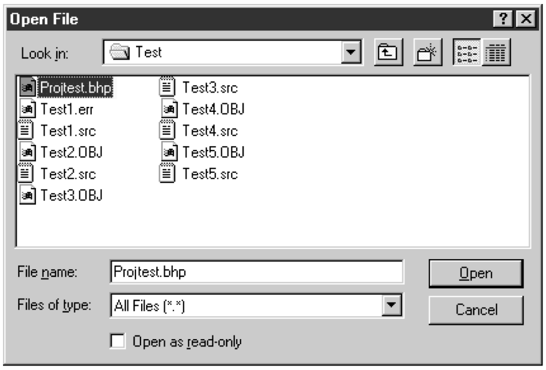
(2) Select an existing project file you want to use (Projtest.bhp in this example), and then click the Open button.

(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.



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

[3]   Add files to a project file

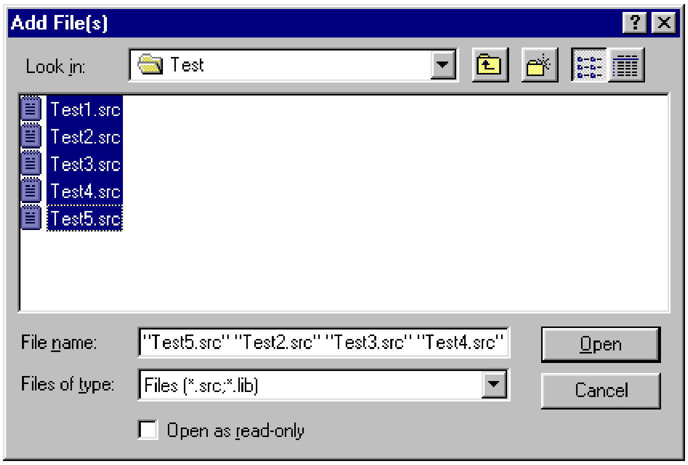
You may add one or more source files and libraries to a project file at a time.

(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.



(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:

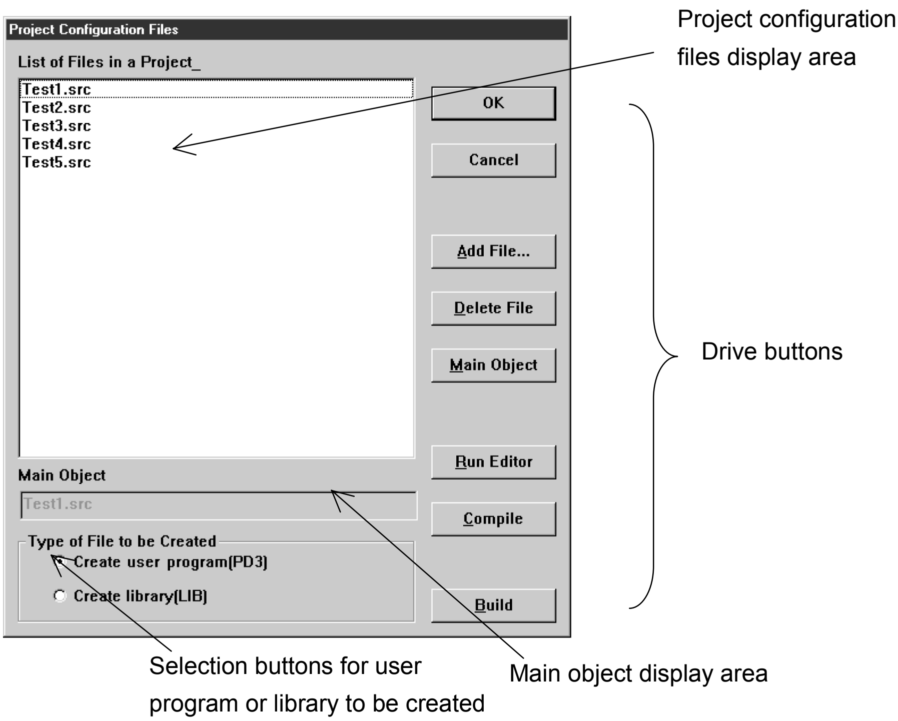
■From the Project menu, choose the Select File command.

■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.



(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 display area*

This 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

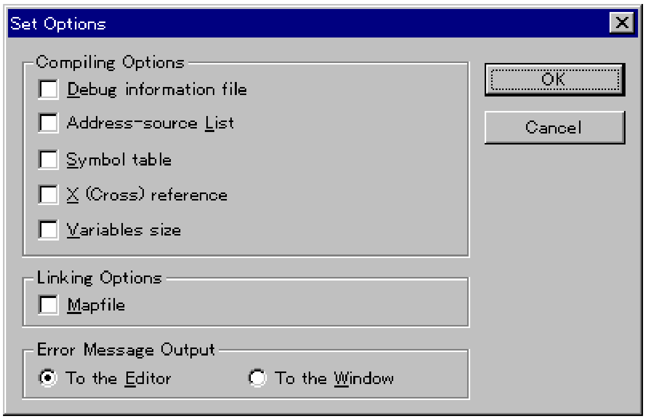
First specify the options and then proceed to the compiling or building process.

[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.



(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):

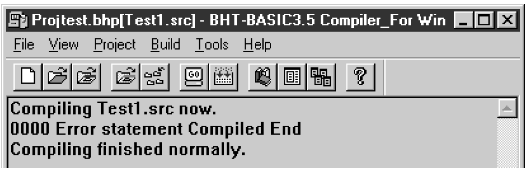
■From the Build menu, choose the Compile command.

■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.



[3] Building

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

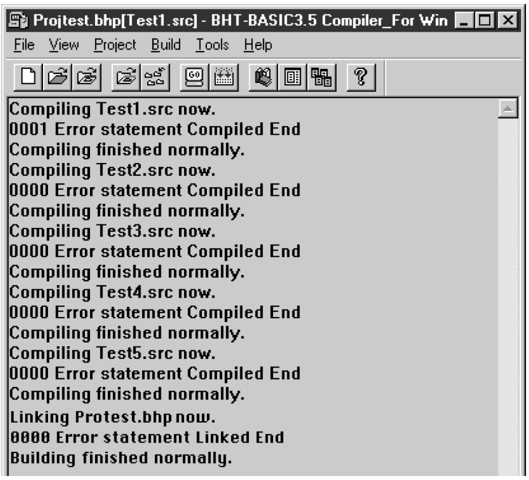
■From the Build menu, choose the Build command.

■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.



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.

(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.

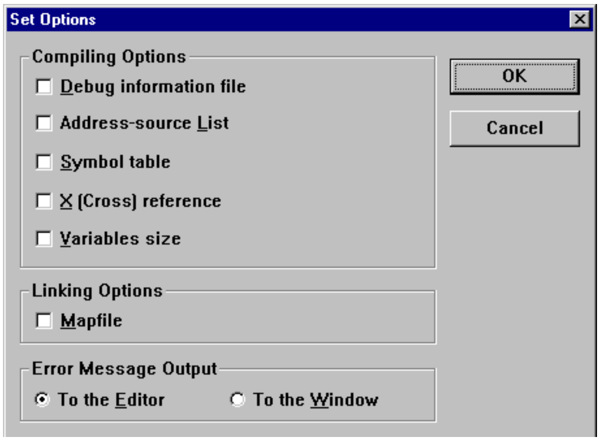
2.4.6 Error Messages and Their Indication onto the Main Window

[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.

The Set Options dialog box appears as shown below.



(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

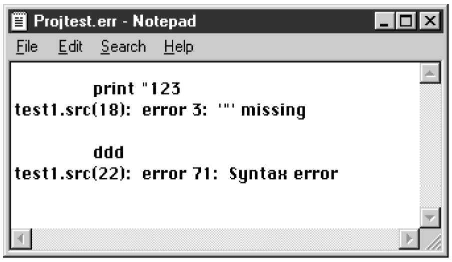
If the Compiler detects a syntax error, it outputs the error message to the XXX.ERR file. For details about the file, refer to Subsection 2.4.9, "Output from the BHT-BASIC 3.5 Compiler."

If the "To the Editor" check box of the Error Message Output is selected in the Set Options dia-log box, the editor will automatically open and show the detected errors. If the "To the Window"

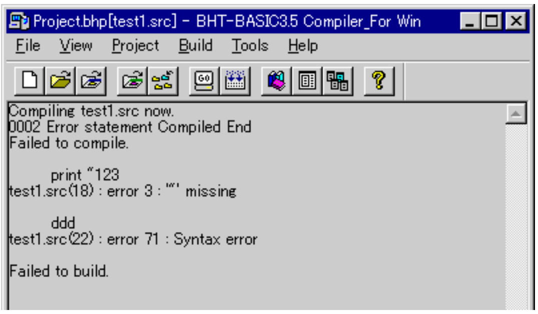
check box is selected, those errors will be outputted to the main window.

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

- Error messages displayed on the editor

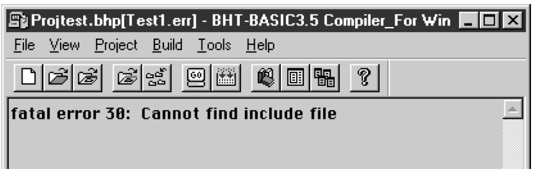


- Error messages displayed on the main window



■Fatal errors

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



■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 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 out-putted.(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	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-defined 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 name 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 WAVE INC. 2001-2002. 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 ←Version of BHT35CPL.DLL
Copyright (C) DENSO WAVE INC. 2001-2002. All rights reserved.
source = C:¥TEST.SRC

   Addr     Line     Statement

   0000     0001     '*******************************************************
   0000     0002     '*
   0000     0014     ON ERROR GOTO ErrorProg
   0003     0015
   0003     0016                     DEFREG vF%=0
   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'
   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

Line number in source program

Source program statement

- **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 ←Version of BHT35CPL.DLL
Copyright (C) DENSO WAVE INC. 2001-2002. All rights reserved.
source = C:¥Test.SRC

 C O M M O N S Y M B O L            ← Symbol table for common variables
 W O R K S Y M B O L                ← Symbol table for register variables
 F%          INPUTERR%     J2%          SEQNO%       SREC%
 SU%         SUBC%         SUBFLAG%     WREC%        X1%

 R E G I S T E R S Y M B O L        ← Symbol table for work variables
 COMF%                     RECNO%

 L A B E L S Y M B O L              ← Symbol table for labels
 AMOUNT      AMOUNTKYIN    CAUTIONB     COMRETRY     DATASET

 L A B E L S Y M B O L              ← Symbol table for user-defined functions
 FNKEYINPUT FNSPAT         FNXCENTER    FNZPAT
```

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 fol-lowing 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
COMMON   SYMBOL
          C%                    2400                      Map for work variables
WORK   SYMBOL
          A                     2900
          B                     2901
          W$                    2A00
                                                          Map for register variables
REGISTER   SYMBOL
          R$                    2E00
                                                          Map for user-defined function
FUNCTION   SYMBOL
          AAA                   003B
                                                          Map for variables and object codes
OBJECT   INFORMATION
                                offset size
          PRC                   0000    0035
          REG                   0035    002F
          PRD                   0064    0047
                                                          Details of object codes
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 (projectname.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 statement 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/NT/2000/XP.

[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]), there-fore, 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 BHTC35W +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	+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-BASIC3.5 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 direc-Tory where the source pro-gram 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 direc-tory where the source program is located) or File YYY.ERR (in the direc-tory 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 direc-Tory 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

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 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 EXECUTE PROGRAM menu.

If you have selected a user program as an auto-start execution program in the SET SYSTEM menu of System Mode, then the BHT will automatically run the program when turned on.

If no user program has been selected as an auto-start execution program, turning the BHT on will transfer the control to Directory Manager that starts a first-loaded one out of user programs (.PD3) loaded in the BHT which will appear on the top of the EXECUTE PROGRAM menu.

For the operating procedure of System Mode, refer to the BHT 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),

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.

or

- the battery voltage level becomes low.

Low battery:	If the voltage level of the rechargeable 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, there-fore, 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 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`.

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 ～ﾝ
• Kanji (2-byte codes) (Full-width characters)	e.g., 漢,ぜ,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 singleline 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 filename.
& (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 includedwithin 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	&HFFFF, &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.

Classification of Variables				Example
String variables		Non-array type		ab3$
		Array type	One-dimensional	e$(10)
			Two-dimensional	gh$(1,3)
Numeric variables	Integer variables	Non-array type		a%
		Array type	One-dimensional	e%(10)
			Two-dimensional	fg%(2,3)
	Real variables	Non-array type		a,bcd
		Array type	One-dimensional	e (10)
			Two-dimensional	fg (2,3)

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 percent-age 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 deleted.

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 deleted 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 power has been turned 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.

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 Real functions Character functions	FN FN FN	*functionname %* *functionname* *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 will occur.

```
-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),...
```

5.5 Scope of Variables

You may scope work variables and register variables to be local or global with the PRIVATE or GLOBAL statement, respectively.

(1) Global variables

A global variable can be accessed by any routine in source files to share information between those routines. Before access to it, you need to declare it with the GLOBAL statement.

(2) Local variables

A local variable can only be accessed by any routine in a source file where it is defined.

Before access to it, you need to declare it with the PRIVATE statement.

(3) Variables not declared to be global or local

If not declared to be global or local, a variable is closed in each file where it is defined.

A variable used inside the FUNCTION or SUB function without declaration is available only within a function where it is defined.

You may also share variables between user programs when one program chains to another by declaring variables to be common with the COMMON statement.

5.5.1 Global Variables

A global variable can be shared between source files in a program. In each file where you want to use a particular global variable, write GLOBAL preceding a desired variable name or DEFREG statement.

```
(Example) GLOBAL aaa%
          GLOBAL bbb$[10]
          GLOBAL ccc$(5,3)[30]
          GLOBAL DEFREG ddd
          GLOBAL DEFREG eee%(5)
```

(Example 1) To share the variable aa% between Files 1 and 2, define aa% by using the GLOBAL statement in each file as follows:

File 1	File 2
GLOBAL aa%	GLOBAL aa%

Before access to a global variable, you should define it.

If used inside the SUB or FUNCTION function in the same file where the global variable is defined, the variable will also have the same value.

(Example 2) The variable aa% defined by the GLOBAL statement will have the same value as aa% within the FUNCTION.

File 1

```
GLOBAL aa%
DECLARE SUB printaa(x)
FUNCTION addaa(x)
   addaa=aa%+x
END FUNCTION
aa%=2
print addaa(2)
printaa(2)
```

File 2

```
GLOBAL aa%
   SUB printaa(x)
   print aa%+x
END SUB
```

If you link Files 1 and 2 above into a program file, the variable aa% used in those files will have the same value.

■If a same name variable is used in one file where it is declared to be global and in the other file where it is not declared

In those files where the variable is declared to be global by the GLOBAL statement, all of those variables will have the same value. In a file where the variable is not declared, the variable is available only in each file.

(Example) If in each of Files 1 and 2 the variable aa% is declared by the GLOBAL statement and in File 3 the variable aa% is not declared:

File 1

```
GLOBAL aa%[50]
```

File 2

```
GLOBAL aa%[50]
```

File 3

```
PRIVATE aa%[50]
```

If you link Files 1, 2, and 3 above into a program file, the variables aa% in Files 1 and 2 will have the same value and aa% in File 3 will be treated as a variable different from those in Files 1 and 2.

5.5.2 Local Variables

A local variable can be accessed only in a file where it is defined. Write PRIVATE preceding a desired variable name or DEFREG statement.

```
(Example)  PRIVATE aaa%
           PRIVATE bbb$[10]
           PRIVATE ccc$(5,3)[30]
           PRIVATE DEFREG ddd
           PRIVATE DEFREG eee%(5)
```

Before access to a local variable, you should define it.

If used inside more than one SUB or FUNCTION function in the same file where the local variable is defined, all of those variables will also have the same value.

```
(Example)   PRIVATE aa%
           FUNCTION addaa(x)
            addaa=aa%+x
           END FUNCTION
           SUB printaa(x)
            print aa%+x
           END SUB
           aa%=2
           print addaa(2)
           printaa(2)
```

In the above example, the variable aa% used in "addaa" and "printaa" will have the same value.

■Variables with overlapping scope

If your program has a global variable and a local variable with the same name, in those files where the variable is declared with the GLOBAL statement, those variables will be treated as the same; in a file where the variable is declared with the PRIVATE variable, the variable is available only in that file.

(Example) If in each of Files 1 and 2 the variable aa% is declared by the GLOBAL statement but in File 3 it is not declared by the GLOBAL statement:

File 1	File 2	File 3
GLOBAL aa%[50]	GLOBAL aa%[50]	PRIVATE aa%[50]

If you link Files 1, 2, and 3 above into a program file, the variables aa% in Files 1 and 2 will have the same value and aa% in File 3 will be treated as a variable different from those in Files 1 and 2.

5.5.3 Variables Not Declared to be Global or Local

If not declared to be global or local, a variable is closed in each file where it is defined. A variable used inside the FUNCTION or SUB function without declaration is available only within a function where it is defined.

```
(Example)  FUNCTION addaa(x)
            addaa=aa%+x
           END FUNCTION
           SUB printaa(x)
            print aa%+x
           END SUB
           aa%=2
           print addaa(2)
           printaa(2)
```

In the above example, all variables aa% used in "addaa," "printaa," and others will be treated as different ones.

5.5.4 Common Variables

A common variable should be declared in a main object beforehand. To share the common variable by files other than the main object, you need to declare it with the `COMMON` statement in each file where the common variable should be available.

<u>File 1</u>

```
DECLARE SUB printaa(x)
COMMON a%
a%=2
printaa(5)
```

<u>File 2</u>

```
COMMON a%
SUB printaa(x)
print a%+x
SUB
```

To use a% as a common variable in Files 1 and 2, define the variable with the `COMMON` statement in each file.

If a common variable declared with the `COMMON` statement is used within the `SUB` or FUNCTION function in a file where the variable is defined, then the common variable will have the same value.

```
(Example)  COMMON aa%
           FUNCTION addaa(x)
            addaa=aa%+x
           END FUNCTION
           SUB printaa(x)
            print aa%+x
           END SUB
           aa%=2
           print addaa(2)
           printaa(2)
```

In the above example, variables aa% used in "addaa" and "printaa" will be treated as same one.

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 string 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] Screen mode and font size

Listed below are the fonts available on BHT.

Screen mode	Font size	Letter type	Character enlargement	Dots (W x H)	Chars x Lines
Single-byte ANK* mode	Standard-size	ANK chars	Regular	6×8	21×8
			Double-width	12×8	10×8
	Small-size	ANK chars	Regular	6×6	21×10
			Double-width	12×6	10×10
Two-byte Kanji mode	Standard-size	Full-width	Regular	16×16	8×4
			Double-width	32×16	4×4
		Half-width	Regular	8×16	16×4
			Double-width	16×16	8×4
	Small-size	Full-width	Regular	12×12	10×5
			Double-width	24×12	5×5
		Half-width	Regular	6×12	21×5
			Double-width	12 ×12	10×5

*ANK: Alphanumerics and Katakana

■Screen mode

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

The 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.

■Font size

The standard- and small-size fonts may be displayed.

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 files.

Switching the screen mode and font size

You may switch the screen mode by using the `SCREEN` statement (*`displaymode`* parameter). Refer to Chapter 14, SCREEN.

You may switch the font size by using the `OUT` statement (port &h6080). Refer to Chapter 14, OUT and Appendix D, "I/O Ports."

[2] Character attributes (Reverse font and enlargement attribute)

■Reverse font attribute
Characters may be reversed (highlighted).

■Enlargement attribute
Characters may be displayed in regular-size and double-width as listed in [1].

Switching the character attributes
You may switch the reverse font attribute and enlargement attribute by using the `SCREEN` statement (*`charaattribute`* parameter). Refer to Chapter 14, SCREEN.

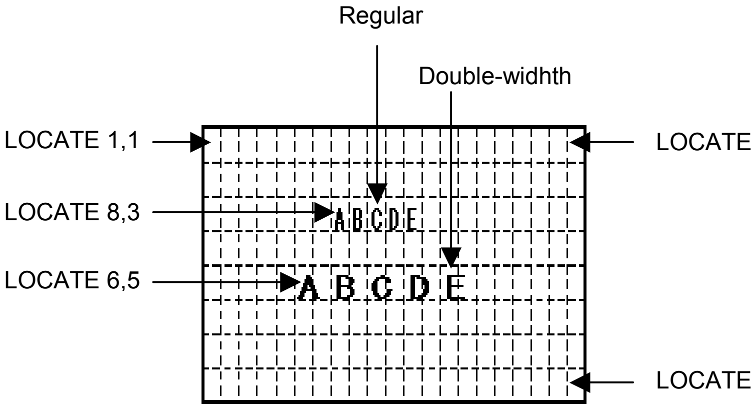
7.1.2 Coordinates on the LCD

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.

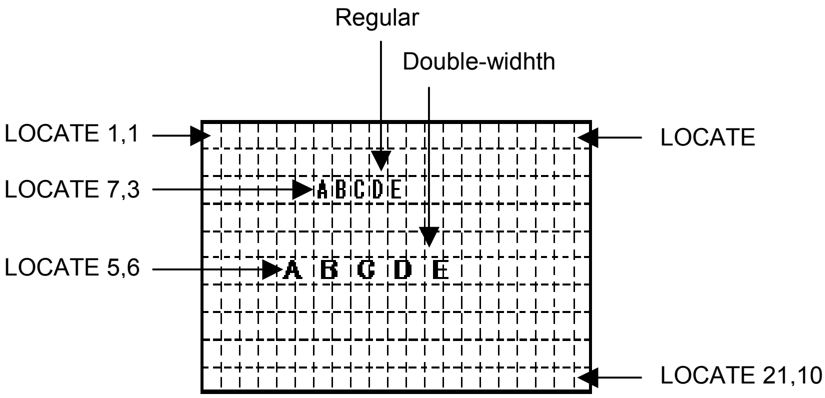
The coordinates will differ depending upon the screen mode and font size.

<u>Single-byte ANK mode</u>

• Standard-size font



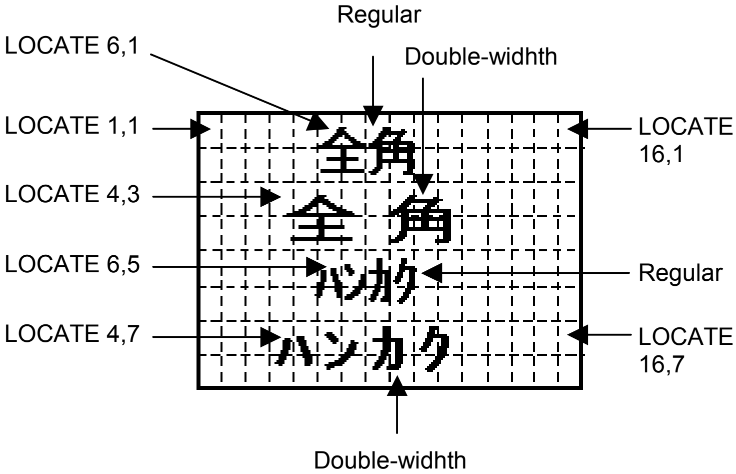
• Small-size font

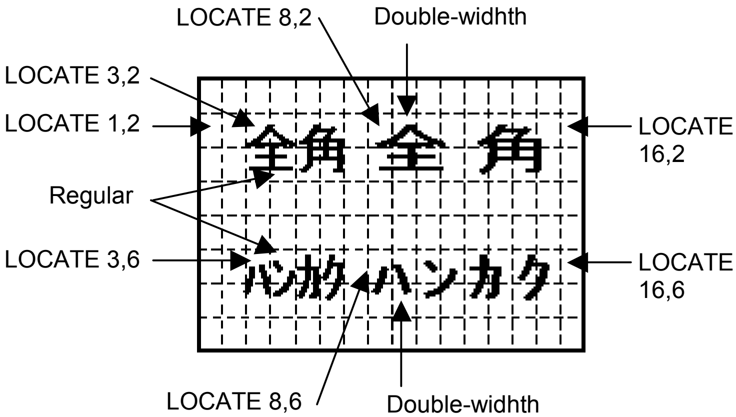


Two-byte Kanji Mode

• Standard-size font

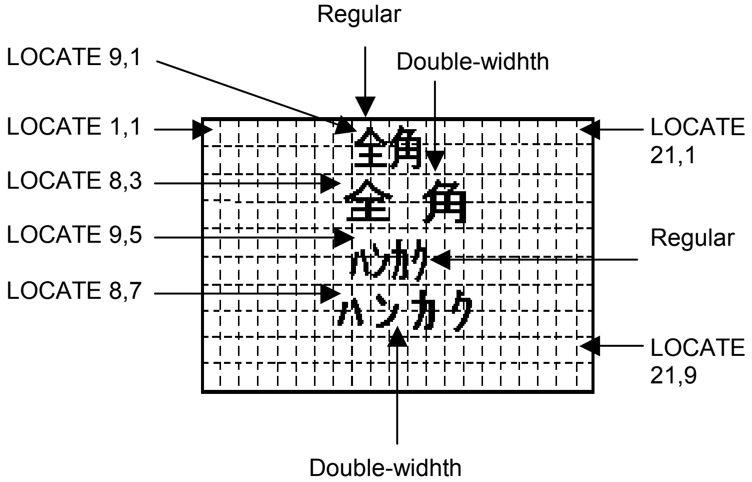
Be careful about the specification of line numbers in figures below. A single column shown below represents an area for a half-width character; Double columns represent an area for a full-width character.

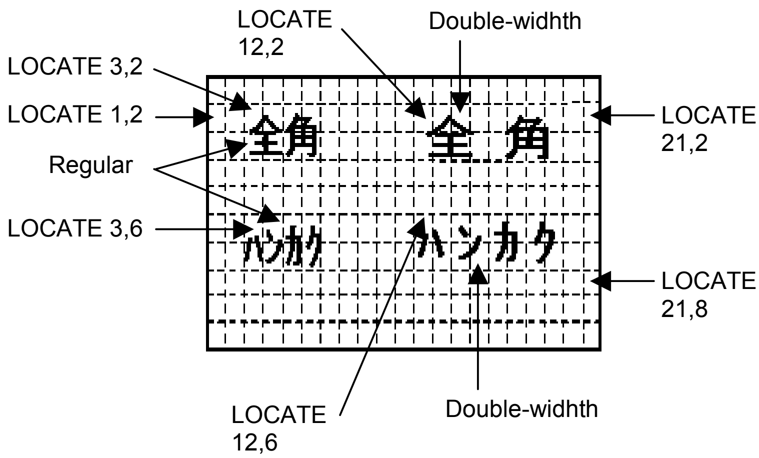




• Small-size font

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 full-width character.





7.1.3 Dot Patterns of Fonts

■**Character fonts**

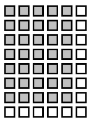
In the figures below, " " shows a display area for characters. Any character is displayed within a set of the display areas.

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

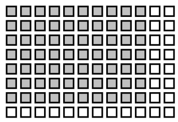
Single-byte ANK mode

• Standard-size font

6 x 8 dots (Regular)

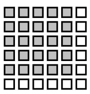


12 x 8 dots (Double-width)

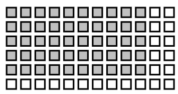


• Small-size font

6 x 6 dots (Regular)



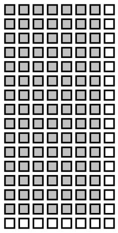
12 x 6 dots (Double-width)



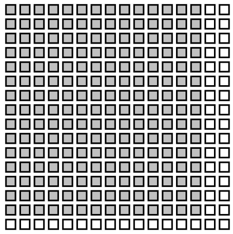
Two-byte Kanji Mode

• Standard-size font

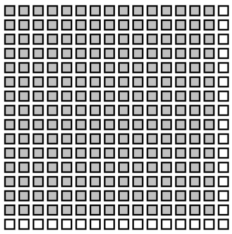
Half-width Kanji
8 x 16 dots (Regular)



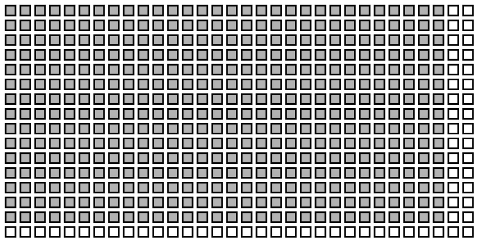
Half-width Kanji
16 x 16 dots (Double-width)



Full-width Kanji
16 x 16 dots (Regular)

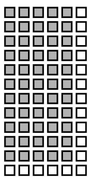


Full-width Kanji
32 x 16 dots (Double-width)

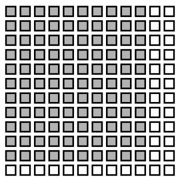


• Small-size font

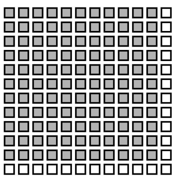
Half-width Kanji
6 x 12 dots (Regular)



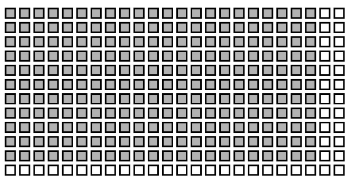
Half-width Kanji
12 x 12 dots (Double-width)



Full-width Kanji
12 x 12 dots (Regular)



Full-width Kanji
24 x 12 dots (Double-width)



■Cursor shape

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

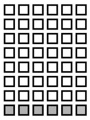
You may define and load the desired cursor shape with the APLOAD or `KPLOAD` statement `and then` specify the user-defined cursor with the `LOCATE` statement. If the double-width character size is specified, the cursor will be displayed in double width.

<u>Single-byte ANK mode</u>

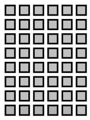
• Standard-size font (6 x 8 dots)

In regular size

Underline cursor



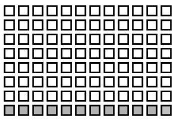
Full block cursor



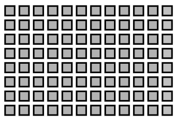
Invisible

In regular size

Underline cursor



Full block cursor

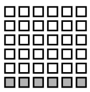


Invisible

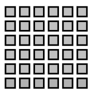
• Small-size font (6 x 6 dots)

In regular size

Underline cursor



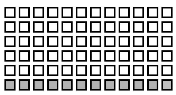
Full block cursor



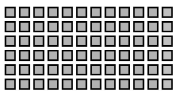
Invisible

In regular size

Underline cursor



Full block cursor



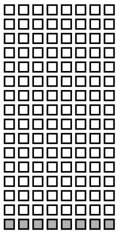
Invisible

Two-byte Kanji Mode

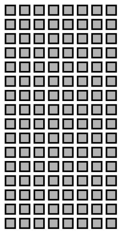
• Standard-size font (8 x 16 dots)

In regular size

Underline cursor



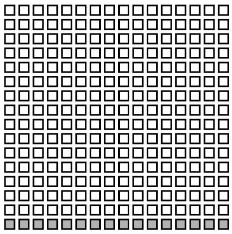
Full block cursor



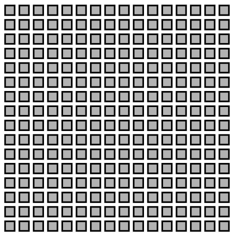
Invisible

In regular size

Underline cursor



Full block cursor

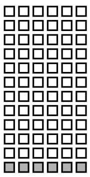


Invisible

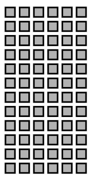
• Small-size font (6 x 12 dots)

In regular size

Underline cursor



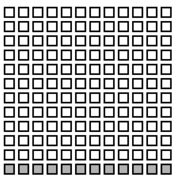
Full block cursor



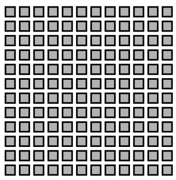
Invisible

In regular size

Underline cursor



Full block cursor



Invisible

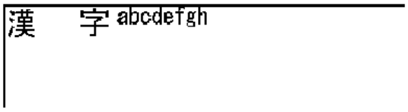
7.1.4 Mixed Display of Different Screen Modes, Font Sizes, and/or Character Enlargement Sizes

[1] ANK Mode and Kanji Mode Together in One Line

ANK characters and Kanji characters may display together in the same line on the LCD screen as shown below.

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

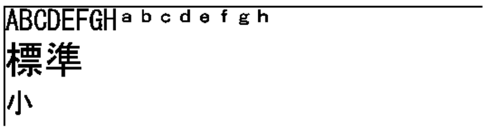
If the display data is outputted to the same location more than one time as shown in the above program, the BHT overwrites the old data with new data.



[2] Standard-Size and Small-Size Fonts Together on the Same Screen

Standard-size and small-size fonts of ANK characters and Kanji characters (both full-width and half-width) may display together on the same screen as shown below.

```
CLS
OUT &h6080, 0                  'Select standard-size font
SCREEN 0                       'Regular-size in ANK mode
PRINT "ABCDEFGH";
OUT &h6080, 1                  'Select small-size font
PRINT "abcdefgh";
OUT &h6080, 0                  'Select standard-size font
SCREEN 1                       'Regular-size in Kanji
mode
LOCATE 1,2:PRINT "標準"
OUT &h6080, 1                  'Select small-size font
PRINT "小";
```



[3] Regular-Size and Double-Width Characters Together on the Same Screen

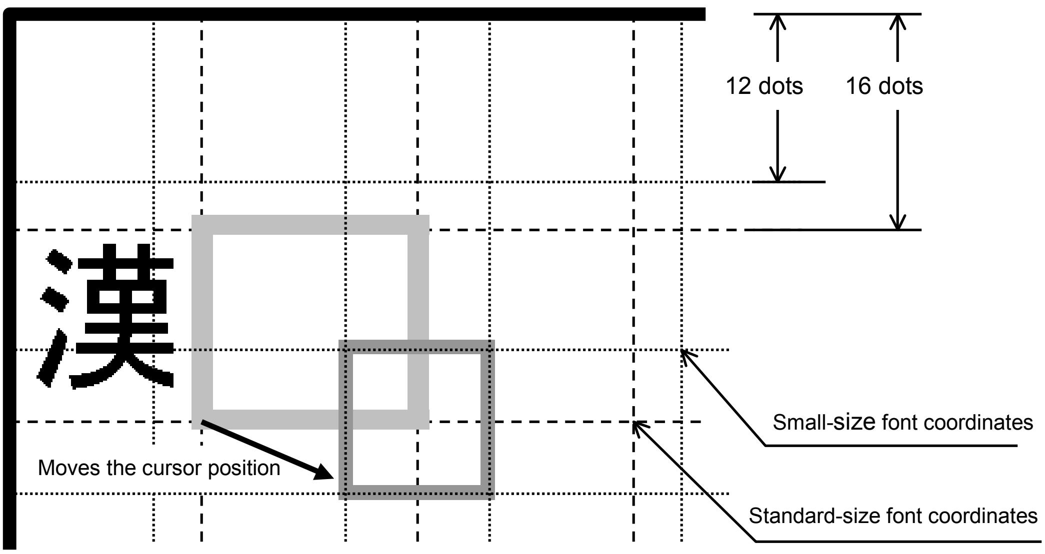
The regular-size and double-width characters may display together on the same screen as shown below.

```
CLS
OUT &h6080, 0                          'Select standard-size font
SCREEN 0,0  : PRINT "ANK"               'Regular-size in ANK mode
SCREEN 0,2  : PRINT "ANK"               'Double-width in ANK mode
SCREEN 1,0  : PRINT "漢字"              'Regular-size in Kanji mode
SCREEN 1,2  : PRINT "漢字"              'Double-width in Kanji mode

LOCATE 1,1
OUT &h6080, 1                          'Select small-size font
SCREEN 0,0  : LOCATE 14  : PRINT "ANK"  'Regular-size in ANK mode
SCREEN 0,2  : LOCATE 14  : PRINT "ANK"  'Double-width in ANK mode
SCREEN 1,0  : LOCATE 14  : PRINT "漢字" 'Regular-size in Kanji mode
SCREEN 1,2  : LOCATE 14  : PRINT "漢字" 'Double-width in Kanji mode
```

■Switching the screen font from the standard-size to small-size

The coordinates on which standard-size fonts are displayed and one on which small-size fonts are displayed are different from each other.

If the screen font is switched from the standard-size to small-size, then the cursor will move from the current position to the nearest lower rightward position on the small-size font coordinates.



7.1.5 Displaying User-defined Characters

■Loading a user-defined font

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

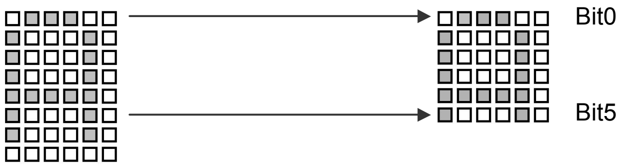
The `APLOAD` statement is capable of loading up to 32 single-byte ANK fonts to be displayed in the single-byte ANK mode.

The `KPLOAD` statement is capable of loading up to 128 two-byte Kanji fonts in full width to be displayed in the two-byte Kanji mode.

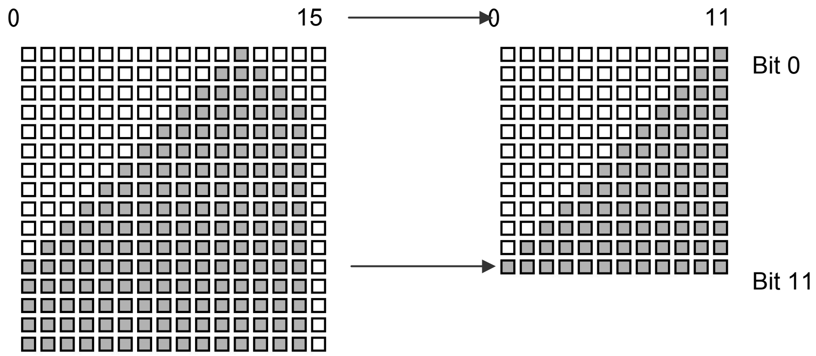
■Enlarging/condensing defined font patterns

If the double-width is specified, the Interpreter will enlarge user-defined font patterns for display.

If the small-size font is specified for font patterns loaded by the `APLOAD` statement, then the Interpreter will use a total of 6 bits (bit 0 to 5) each on the 1st to 6th elements and ignores the bits 6 to 7, as shown below.



If the small-size font is specified for font patterns loaded by the `KPLOAD` statement, then the Interpreter will use 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, as shown below.



7.1.6 VRAM

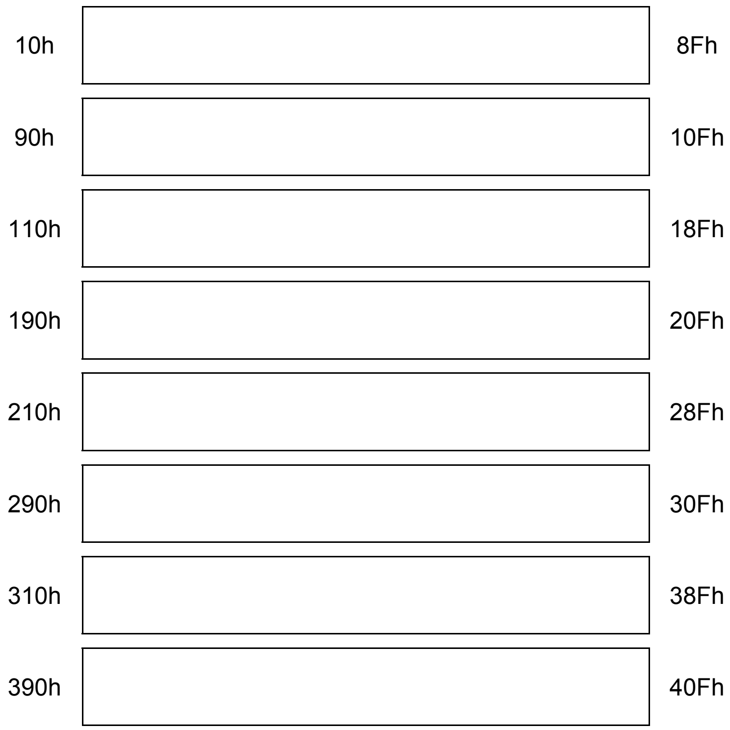
The `INP` function may read the VRAM data. The `OUT` statement writes data into the VRAM so that graphics may be displayed on the LCD dotwise.

■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-8000	10h to 40Fh

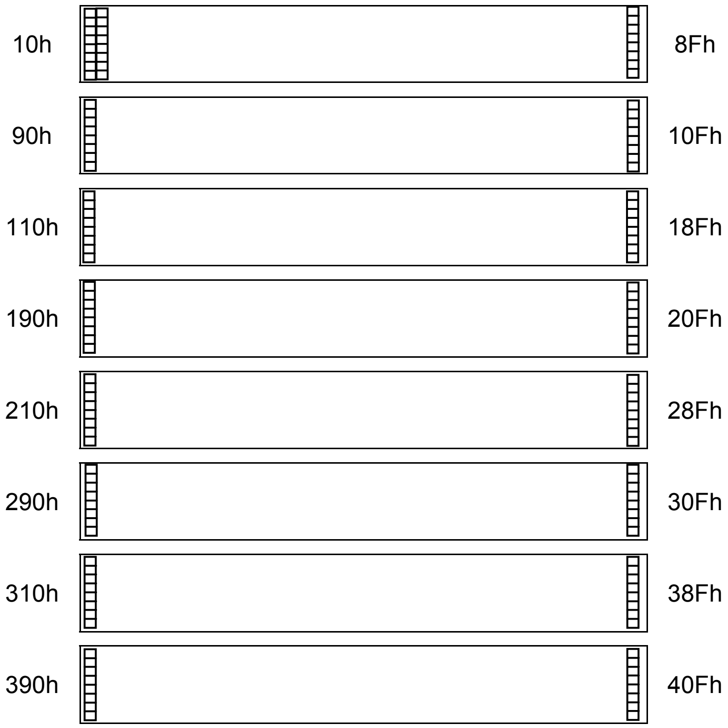
Port numbering system counts, starting from the top left corner of the LCD to the right bottom corner.



■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) in the `OUT` statement or `INP` function. If the bit is 1, the corresponding dot on the LCD will come ON.

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



7.1.7 Displaying the System

The BHT-8000 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.

■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.

Refer to Appendix C2, "National Character Sets."

■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

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, it will become equal to the size of the half-width characters in each mode.

7.2 Input from the Keyboard

7.2.1 Alphabet Entry

In addition to the numeric entry from the keypad, the BHT-8000 supports software keyboard 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   'Switch to the numeric entry system*
OUT &h60B0,1   'Switch to the alphanumeric entry system
```

*Selected when the BHT-8000 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-8000 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   'Switch to the numeric entry mode
OUT &h60B1,1   'Switch 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** icon 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 high-lighted 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, space, y, z
0	\\+, -, *,
.	/, $, %, comma (,)

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.

7.2.2 Other Facilities for the Keyboard

[1] Auto-repeat

The keys on the BHT series are not auto-repeat.

[2] Shift key

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, the SF icon appears 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 turning on the BHT. (For the adjustment of the beeper volume, refer to the BHT 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 indicator LED in green. Turns on the indicator LED in red. Turns off the indicator 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 & touch key 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

* 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.

The table below lists some examples.

`WAIT` Statement	I/O Devices
`WAIT 0,&h01`	Keyboard buffer & touch key buffer status
`WAIT 0,&h02`	Barcode 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

* The `WAIT` statement 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 keyboard buffer & touch key buffer and the barcode 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

CONTENTS

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*	
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 user area is located at drives A and B. Note that drive B in the BHT-8000 is provided for ensuring the compatibility with conventional BHT series.

The memory space available for data files is (Memory space on drive A - 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.

The filename`.extension` may 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.

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 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-100 receives a file with the XFILE statement, it will automatically set the maxi-mum number of registrable records to 32,767 for that file.

  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

During downloading, 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.

- Longer beep than usual
- Keys do not function
- Bar codes cannot read
- Refreshing of the LCD screen is delayed
- Data cannot be received
- `TIMEA`, `TIMEB`, or `TIMEC` operation 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.

If the auto power-off function is enabled (refer to the `POWER` statement in Chapter 14) in the BHT-8000, the system may automatically defragment the drive at the execution of the auto power-off function. It will take approx. 10 seconds. During defragmentation, a progress bar graph will be displayed. Until the completion of defragmentation, the battery should not be removed from the BHT-8000.

For details about defragmentation with `OUT` statement, refer to Appendix D, "I/O Ports."

8.2.6 About Drives

The BHT-8000 has logical drives.

Drive B is provided for ensuring compatibility with other BHT series.

If you specify drive name "B:" preceding a filename.extension and open an existing file, then the BHT will open the file as a read-only file. Executing the PUT statement to the read-only file will result in a run-time error (43h).

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*1 EAN-8 UPC-A*1 UPC-E	No national flag specified.
Interleaved 2of5 (ITF)		No length of read data specified. No check digit.
Standard 2of5 (STF)		No length of read data specified. No check digit. Short format of the start/stop characters supported.
Codabar (NW-7)		No length of read data specified. No check digit. No start/stop character.
Code 39		No length of read data specified. No check digit.
Code 93		No length of read data specified.
Code 128 (EAN-128)*2		No length of read data specified.

*1 Reading wide bars

EAN-13 and UPC-A bar codes may be wider than the readable area of the bar-code reading window.

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.)

*2 Specifying Code 128 makes it possible to read not only Code 128 but also 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, NW-7, Code 39, and STF)
- Length of read data
- Start/stop character (only for NW-7 and STF)
- Country code represented by flag characters (only for universal product codes)
- Supplemental code (only for universal product codes)

■**Barcode Buffer**

The barcode buffer stores the inputted bar code data.

The barcode buffer will be occupied by one operator entry job and can contain up to 99 characters.

You can check whether the barcode buffer stores 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 mark (denoting 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 and two bytes for the data length.

■Multiple Code Reading

You may activate the multiple code reading feature which reads more than one code type while automatically identifying them. To do it, you should designate desired code types following the CODE in the OPEN "BAR:" statement.

■Read Mode of the Trigger Switch

The trigger switch function is assigned to the magic keys M3 and M4 by default. 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 as listed below.

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.

■Generating a Check Digit of Bar Code Data

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 Indicator LED and Beeper (Vibrator) for of Successful Reading

By using the `OPEN "BAR:"` statement, you can control:

- whether the indicator LED should light in green or not (Default: Light in green)
- whether the beeper should beep or not (Default: No beep)

(The BHT-8000 may control the vibrator also.)

when a bar code is read successfully. For detailed specification of the `OPEN "BAR:"` statement, refer to Chapter 14.

Controlling the indicator LED

If you have activated the indicator 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 indicator 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 indicator 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 have activated the beeper in the `OPEN "BAR:"` statement, the BHT will beep when it reads a bar code successfully.

You may choose beeping only, vibrating only, or beeping & vibrating on the LCD screen or by setting the output port in the `OUT` statement.

This feature is used to sound the beeper or operate the vibrator the moment the BHT-8000 reads a bar code successfully.

8.4 Communications Device

8.4.1 Overview

The available communications interface in BHT is as follows.

- IrDA interface
- Direct-connect interface
- Bluetooth interface (For BHTs with Bluetooth communications device)

For the Bluetooth interface, refer to Chapter 18.

8.4.2 Hardware Required for Data Communications

The following hardware is required for communications between the BHT and the host computer:

- Optical communications unit (CU) and its interface cable

or

- Direct-connect interface cable

For the communications specifications, refer to the BHT 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."

8.4.3 Programming for Data Communications

Setting the Communications Parameters

Use the `OPEN "COM:"` statement to set the communications parameters.

■For IrDA interface

Communications Parameters	Effective Setting	Default
Transmission speed (bps)	115200,57600,38400,19200,9600,2400	9600

Parameters other than the transmission speed are fixed (Parity = None, Character length = 8bits, Stop bit length = 1 bit), since the physical layer of the IrDA interface complies with the IrDA-SIR 1.2.

■For direct-connect interface

Communications Parameters	Effective Setting	Default
Transmission speed (bps)	115200,57600,38400,19200,9600, 4800,2400, 1200,600,300	9600
Parity*	None, even, or odd	None
Character length*	7 or 8 bits	8 bits
Stop bit length*	1 or 2 bits	1 bit

* 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.

8.4.4 Overview of Communications Protocols

The BHT supports two communications protocols—BHT-protocol and BHT-Ir protocol for file transmission. Using the XFILE statement, the BHT may upload or download a file according to either of these protocols.

[1] BHT-protocol

This protocol may be used also in System Mode.

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 transmits files according to the BHT-protocol, each field length should be a maximum of 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

In addition to the BHT-protocol, the BHT supports the BHT-Ir protocol.

If you select the BHT-Ir protocol by using the `OUT` statement (Port &h6060) or in System Mode, you can upload or download a data file with the `XFILE` statement.

The BHT-Ir protocol may be used also in System Mode.

For the communications specifications of the BHT-Ir 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-Ir protocol, you cannot change the values of the headers and terminator with the protocol functions in BHT-BASIC.

8.4.5 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, then 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, 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, 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."

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 trap-ping.

- 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 handles 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 instruction 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=0034
```

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 subroutines 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 ***"
                PRINT ERR,HEX$(ERL)
                RESUME NEXT
```

If a run-time error occurs in the main routine, the above program executes the error-handling routine specified by the 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 magic keys M1 and M2 (these keys are numbered 30 and 31, respectively).

```
                ON KEY (30)GOSUB sub1
                ON KEY (31)GOSUB sub2
                KEY (30)ON
                KEY (31)ON
                :
                :
                (Main routine)
                :
                :
                END
sub1
                (Event-handling routine 1)
                RETURN
sub2
                (Event-handling routine 2)
                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 timer 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 value less than 10 seconds is set to a `TIMEA, TIMEB,` or `TIMEC` function so that the returned value is a nonzero.
- 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 being on
- When any key is held down.
- When the backlight is on (except when the backlight is kept on).
- When the beeper is beeping.
- When the vibrator is working.
- 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 pro-gram (user program) when the BHT is turned off, and then resumes it when the BHT is turned 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 and turn 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 statements:

- 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

If the output voltage of the battery cartridge drops below a specified lower level limit when the BHT is in operation, then 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 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 User’s Manual.

12.2 Prohibited Simultaneous Operation of the Beeper, Illumination LED, and LCD Backlight

The BHT is so designed that the beeper (and vibrator), 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 (and vibrator) has the highest priority, the illumination LED has the next priority, and the LCD backlight has the lowest priority.

12.3 Wakeup Function

The wakeup function allows you to turn the BHT on at the wakeup time (of the system clock) specified in user programs.

To set the wakeup time, use the `TIME$` function as follows:

(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 preset wakeup time, use the `TIME$` function as follows:

(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.

TIP

If you set or retrieve the system time or wakeup time by using the TIME$ function, then 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, it means that the BHT is initiated by the wakeup function and if 0, it means that it is initiated by the PW key.

12.4 Remote Wakeup Function

[1] Outline

The remote wakeup function allows you to wake up the BHT from a remote location so as to 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-8000)
- CU interface cable

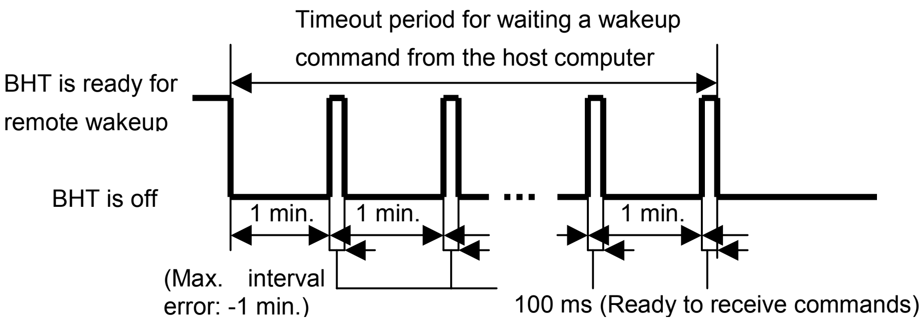
NOTE: If you will not use the remote wakeup function, it is recommended that it be deactivated. This is because activating the remote wakeup function will periodically run the CPU for the specified time length during the wakeup effective hours (timeout period) so that the BHT will consume more power than that with the remote wakeup function deactivated.

[2] Remote wakeup operation

■About BHT internal operation enabling remote wakeup

If the BHT is turned off normally* with the remote wakeup function activated, then it will become ready to receive commands from the host computer at the timing shown below during the specified timeout period. During this operation, nothing appears on the LCD.

(*"Turned off normally" refers to turned-off with the PW key, with the auto power-off feature, or with END, POWER OFF, or POWER 0 statement. If the BHT is shut down due to low battery or no battery loaded, it will no longer become ready for remote wakeup operation.)



Set up the host computer and BHT so that the BHT may receive commands from the host computer at the timing shown above, referring to the typical operation flow given below.

■Configuring the BHT for the remote wakeup

To use the remote wakeup, you need to configure the BHT in System Mode or in user programs as listed below.

For the operating procedure in System Mode, refer to the BHT User's Manual. For that in user programs, refer to "[3] Remote wakeup program."

Items	Set values
Remote wakeup function	Activate
Transmission speed	Match the CU's and host's transmission speed.
Timeout period (Effective hours)	Match the timeout specified in the host's application.

■Typical operation flow

<u>**At the host computer**</u>

(1) Send a "WAKE" character string to the BHT.

(2) Wait for a response from the BHT.

-If the host receives "ACK + 0 + ID":

The host should conduct transactions with the remote wakeup program in the BHT.

-If the host receives "EOT + 1 + ID":

The host should proceed to the corresponding error processing.

If the host receives no response from the BHT for 30 ms, go back to step (1).

(3) Perform steps (1) and (2) repeatedly for 60 seconds or more. If the host receives no response from the BHT during the period, it should proceed to the specified error processing.

Refer to the sample program given below.

sample_e.c

At the BHT

(1) Turn the BHT off and put it on the CU.

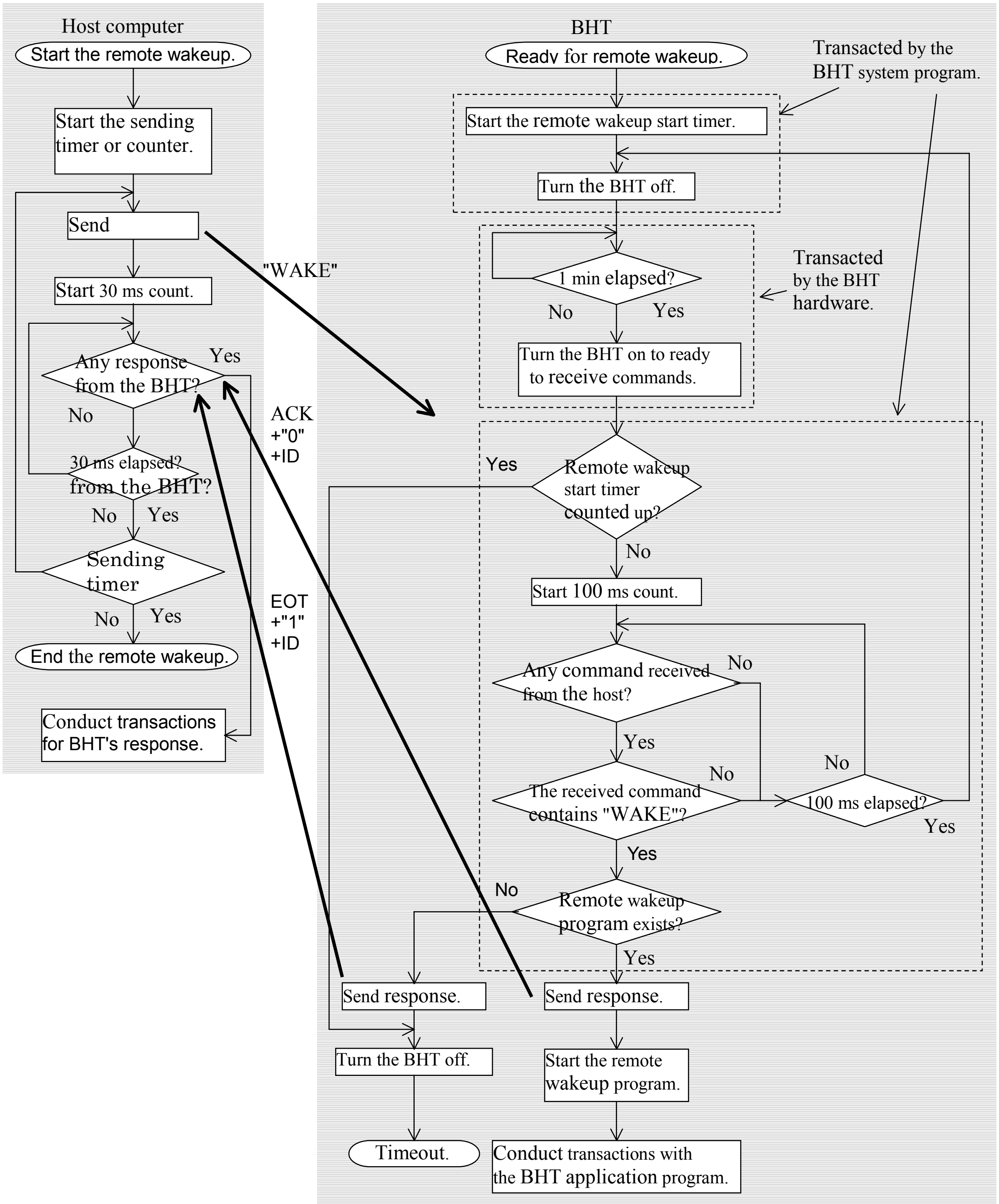
(2) Upon receipt of any data, the BHT will check the data.

If the BHT detects a "WAKE" character string in the data, it will proceed to step (3); if not, it will go back to step (1).

(3) The BHT will send the following response to the host computer depending upon whether or not a remote wakeup program exists in the BHT.

Remote wakeup program	Response message from the BHT	Proceeds to:
Exists	ACK + "0" + ID*	(4)
Not exist	EOT + "1" + ID	(2)

*ID: 6-byte numeric string that refers to the lower 6 digits of the BHT product number.



(4) The BHT will exit from the off state and execute the remote wakeup program developed by the user.

[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, 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 statement beforehand (refer to Appendix D, "I/O Ports"):

(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 5 (115200 bps).

Port No.	Bit No.	R/W	Specifications
60F1h	7-0	R/W	1: 9600 bps2: 19200 bps 3: 38400 bps4: 57600 bps 5: 115200 bps

(3) Set the timeout period for ready-to-receive state

Set the timeout period during which the BHT will be ready to receive a remote wakeup command from the host computer. The default is 12 (hours).

Port No.	Bit No.	R/W	Specifications
60F4h	7-0	R/W	1 to 24 (hours).

(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 extended 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 the termination of the user program.

The remote wakeup activation/deactivation, transmission speed to be applied for remote wakeup, and timeout period for ready-to receive state may be set in System Mode. For details, refer to the BHT 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 extended function SYSTEM.FN3 (Function #1). For details, refer to Chapter 16, "Extended 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.")

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.

■If the dry cells or battery cartridge is unloaded and reloaded when the BHT is ready for remote wakeup

When the BHT is ready to receive remote wakeup commands, unloading and reloading the dry cells or battery cartridge may not retain the ready-to-receive state.

To use the remote wakeup after that, turn the BHT on and off. The BHT will become ready for remote wakeup and the remote wakeup start timer will start counting from the beginning.

Chapter 13
Backlight Function

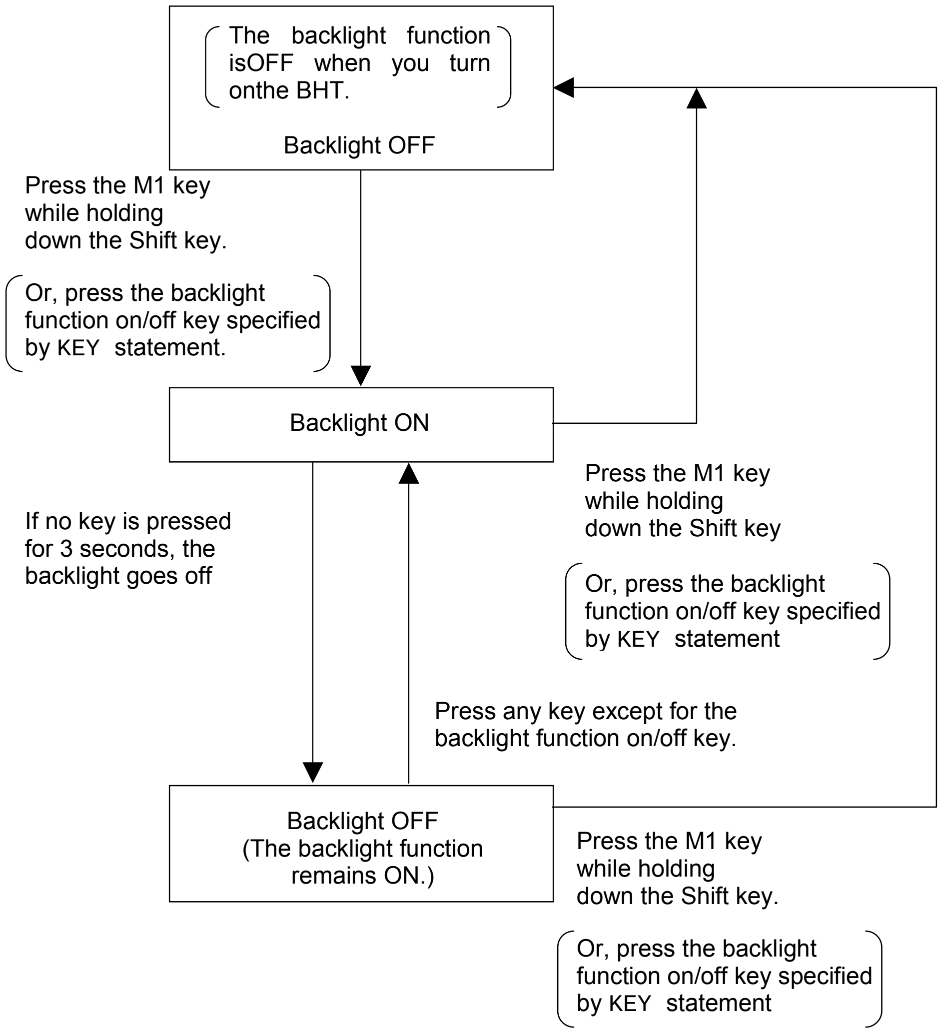
CONTENTS

13.1 Backlight Function

The BHT has a backlight function (LCD backlight and key backlight). Pressing the M1 key 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 an `OUT` statement, you can enable/disable either or both the LCD backlight and key backlight. (Refer to Appendix D, "I/O Ports.")

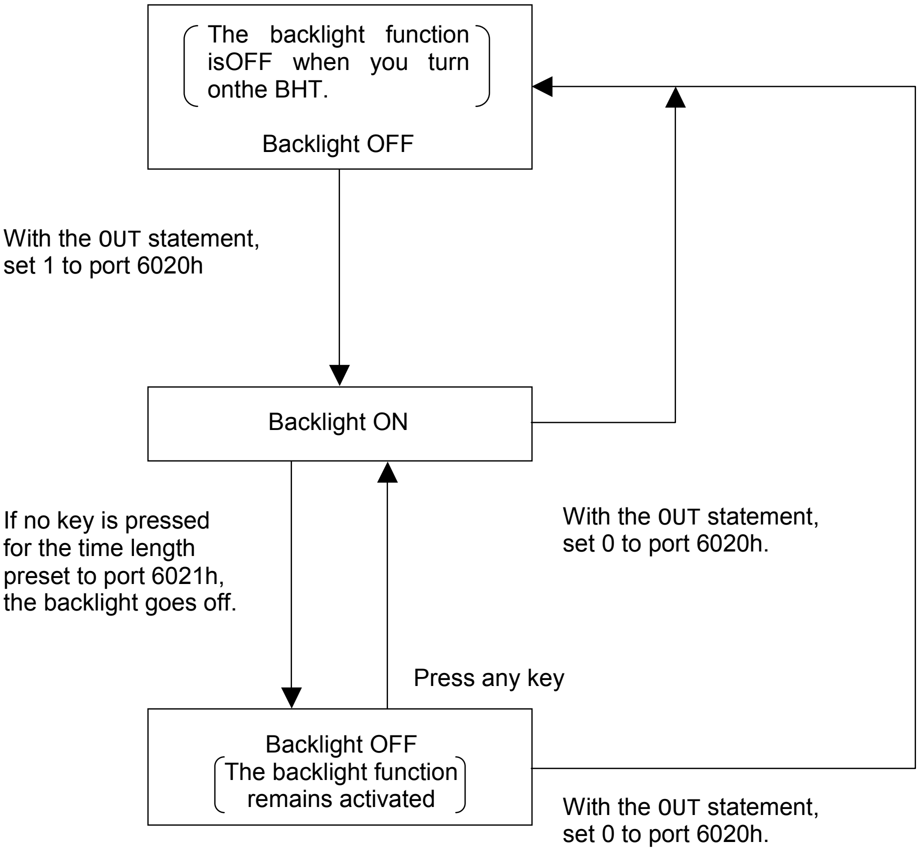
By using a `KEY` statement, you can select the backlight function on/off key instead of the combination of the trigger switch and Shift key, as well as modifying the ON-duration of the backlight. For details about the `KEY` statement, refer to `KEY` in Chapter 14.



Setting 1 to port 6020h with the `OUT` statement activates the backlight function and turns on the backlight. If no key is pressed for the time length preset to port 6021h (default time: 3 seconds), the backlight goes off but the backlight function remains activated.

Setting 0 to port 6020h deactivates the 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.



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.):

```
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; doing so 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 load-ing 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%(11)
DEFREG cp1%(11)
COMMON cp2%(11)
```

  The array variable should be one-dimensional and have at least 6 elements.

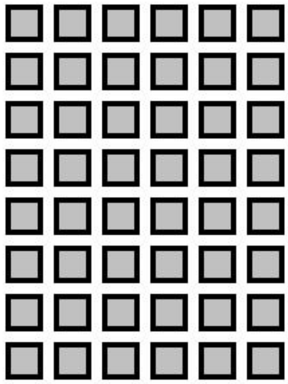
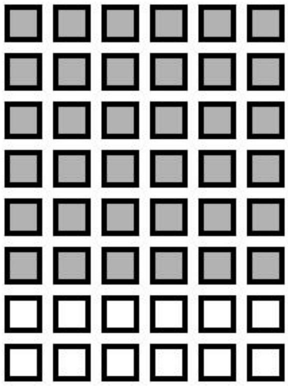
  Each element data should be an integer and stored in the area from the 1st to 6th elements of the array.
- Also when the small-size font or double-width is specified, user-defined fonts loaded by the `APLOAD` will be effective. For those font patterns, refer to Chapter 7, Subsection 7.1.3, "Dot Patterns of Fonts" and Subsection 7.1.5, "Displaying User-defined Characters."

■Loading a user-defined cursor

`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 set 255 to the *cursorarrayname* 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 be as shown below.

Display font	Size (W×H)	No. of elements
Standard-size	6×8 dots LSB  MSB	6
Small-size	6×6 dots LSB  MSB	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%(11)
DEFREG cp1%(11)
COMMON cp2%(11)
```

  The array variable should be one-dimensional and have at least 12 elements.

  Each element data should be an integer and stored in the area from the 1st to 12th 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.
- If the double-width, double-height, or quadruple-size is specified, then user-defined cursors loaded by the `APLOAD` will display in double-width, double-height, or quadruple-size, respectively. For details, refer to Chapter 7, Subsection 7.1.3, "Dot Patterns of Fonts."

Syntax errors:

Error code and message	Meaning
`error 71: Syntax error`	• No *`fontarrayname`* or *`cursorarrayname`* is defined. • *`fontarrayname`* or *`cursorarrayname`* has an array string variable. • *`fontarrayname`* or *`cursorarrayname`* includes parentheses ( ). • *`fontarrayname`* or *`cursorarrayname`* 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`

I/O statement

BEEP

Drives the beeper or vibrator.

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:	2793 Hz* (*Same as that 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*	Frequency	Tone	Statement example
0	698 Hz	Low-pitched	BEEP ,,,0
1	1396 Hz	Medium-pitched	BEEP ,,,1
2	2793 Hz	High-pitched	BEEP ,,,2

Specification of 0, 1, or 2 to *frequency* drives the beeper or vibrator depending upon the settings made on the main adjustment screen of the LCD, beeper, and touch 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 turning on the BHT. (For the adjustment procedure, refer to the BHT User's Manual.)

You may change the beeper volume with the `OUT` statement. (For details, refer to Appendix D, "I/O Ports.")

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.

- Specification of any of 3 through 61 to *`frequency`* deactivates the beeper or vibrator.
- 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.
- 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
```

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.

- [*`drivename`*:] is used in conventional BHT models. In the BHT-100 series, it is merely for the compatibility with their specifications. The *`drivename`* may be `A:` or `B:`, but it will be ignored.

- *`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 *`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:

  Reserve a storage area for a returned string variable by using a variable declaration statement (`DIM, COMMON,` or `DEFREG`). It is not necessary 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 length reserved by a variable declaration statement, then 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]          ' Reserve 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]          ' Reserve 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.

■**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 *`realparameters`* should be equal to that of *`dummyparameters`*, 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 *`realparameters`* 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 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 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`

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*:]*programfilename*". 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*:]*programfilename*" and executes it.

- [*drivename*:] is used in conventional BHT series. In the BHT-100 series, it is merely for the compatibility with their specifications. The *drivename* may be A: or B:, but it will be ignored.
- "[*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 run-time error 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 syntax 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`

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 pro-gram 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 data in drive B

Example:

```
OPEN "master.Dat"AS #1
FIELD #1,20 AS bar$,10 AS ky$
CLFILE #1
CLOSE #1
```

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 sub-sequent `OPEN` statement.
- If no file number is specified, the `CLOSE` statement closes all of the opened data files and device I/O files.
- Specifying an 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`

CLear Screen I/O statement

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 settings made by *`displaymode`* or *`charaattribute`* in the `SCREEN` statement. (For details about *`display-mode`* and *`charaattribute`*, refer to the `SCREEN` statement.)
- This statement turns off the cursor.
- 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.

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 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 sub-script ranging from 0 to 254 for an array variable.
- The total variable data size which can be passed between chained programs is 32 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

CONST

Defines symbolic constants to be replaced with labels.

Syntax:

`CONST` *`constname`* `=` *`expr`*

Parameter:

`constname`

A label, identifier, or string expression of characters consisting of alphanumerics and period (.).

`expr`

A string

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.

I/O statement

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.

- The cursor size depends upon the screen mode (single-byte ANK mode or two-byte Kanji mode), the screen font size (standard-size or small-size), and the character enlargement attribute (regular-size, double-width, double-height, or quadruple-size). For details about the cursor, refer to Chapter 7, Subsection 7.1.3.
- The cursor shape specified by the most recently executed LOCATE statement takes effect.
- After execution of LOCATE ,,0 which makes the cursor invisible, even execution of 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$

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 since the RESTORE can change a location where the READ statement should start reading data.
- 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

User-defined function declarative statement

DECLARE

Declares user-defined function `FUNCTION` or `SUB` externally defined.

Syntax:

Syntax 1 (Defining a numeric `FUNCTION`):

`DECLARE FUNCTION` *`funcname`* `[(`*`dummyparameter`*`[,`*`dummyparameter`*`...])]`

Syntax 2 (Defining a string `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$	String 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-defined 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

DEFine FuNction

User-defined function definition statement (Single-line form)

DEF FN

Names and defines a user-defined function.

Syntax:

Syntax 1 (Defining a numeric function):

DEF FN*functionname*[(*dummyparameter*[,*dummyparameter*...])]=*expression*

Syntax 2 (Defining a string function):

DEF FN*functionname*[(*dummyparameter* [,*dummyparameter*...])] [[*stringlength*]]=*expression*

Syntax 3 (Calling the function):

FN*functionname*[(*realparameter*[,*realparameter* ...])]

Parameter:

functionname

- For numerics

functionname%	Integer function name
functionname	Real function name

- For strings

functionname$	String 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 string 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 *dummyparameter* is used outside `DEF FN` statement or used as a *dummyparameter* of any other function in the same program, then 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 *realparameters* should be equal to that of *dummyparameters*, 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 *realparameters* 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 incontrol 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 *`dummypa-rameter`* 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 `DEF FN` statements to more than 10 levels.)
`0Fh`	String length out of the range (The returned value of the *`stringlength`* exceeds the allow-able 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
```

DEFine FuNction...END DEFine User-defined function definition statement

DEF FN…END DEF

(Block form)

Names and defines a user-defined function.

Syntax:

Syntax 1 (Defining a numeric function):

`DEF FN`*`functionname`*`[(`*`dummyparameter`*`[,`*`dummyparameter`*`...])]`

Syntax 2 (Defining a string function):

`DEF FN`*`functionname`*`[(`*`dummyparameter`*`[,`*`dummyparameter`*`...])][[`*`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 string 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, then 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 user-defined function. The type of *`functionname`* should match that of a return value. If no return value is assigned to *`functionname`*, then the value 0 or a null string will be returned for a numeric function or a string function, respectively.

■Calling a user-defined function

FN*functionname* calls a user-defined function.

- The number of *realparameters* should be equal to that of *dummyparameters*, 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 *realparameters* are passed not by address but by value. (So called "Call-by-value")

Syntax errors:

■When creating a user-defined 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 struc-ture (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 *`dummypa-rameter`* 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
```

DEFine REGister Declarative statement

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 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`

DIMension Memory control statement

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 that 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`

Flow control statement

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 executes, all of the files being opened become closed, and the BHT turns off the power after three seconds from the message indication of the "Program end."

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`

File I/O statement

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 *`fieldwidths`* 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.

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`

Flow control statement

FOR…NEXT

Defines a loop containing statements to be executed a specified number of times.

Syntax:

```
FOR controlvariable = initialvalue TO finalvalue [STEPincrement]
 -
 -
 -
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. Reusing it will cause a run-time error 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.)

User-defined function statement

FUNCTION…END FUNCTION

Names and creates user-defined function `FUNCTION`.

Syntax:

Syntax 1 (Defining a numeric function):

`FUNCTION` *funcname* `[(`*dummyparameter* `[,`*dummyparameter*`...])]`

Syntax 2 (Defining a string 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$	String 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 string 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, then 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, IFTHEN ..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 avail-able 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 user-defined function. The type of *`funcname`* should match that of a return value. If no return value is assigned to *`funcname`*, then the value 0 or a null string will be returned for a numeric function or a string function, respectively.

■Calling a user-defined function

`funcname` calls the function.

- The number of *`realparameters`* should be equal to that of *`dummyparameters`*, 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 *realparameters* 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 def-inition of the FUNCTION function or declaration of the FUNCTION by the DECLARE statement in your source program.

Syntax errors:

■When programming a user-defined 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 FUNC-TION, 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 96: Incomplete control structure(FUNC-TION...END FUNCTION)`	`END FUNCTION` is missing.
`error 97: Cannot use FUNCTION in control structure`	The `FUNCTION…END FUNCTION` statement is defined in other blockstructured 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

Read 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:

Syntax 1:

```
GLOBAL varname [,varname...]
```

Syntax 2:

```
GLOBAL DEFREG registerdefinition [,registerdefinition...]
```

Parameter:

```
varname
   numericvar [(subscript)]
   stringvar [(subscript)[[stringlength]]]
registerdefinition
   non-arraynumericvar [=numericconstant]
   arraynumericvar(subscript) [=numericinitialvaluedefinition]
   non-arraystringvar[[stringlength]][=stringconstant]
   arraystringvar(subscript)[[stringlength]][=stringinitialvaluedef
   inition]
   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 which indicates the number of characters.

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 variable 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, PRI-VATE, 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 pre-vent 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`
`07h`	Insufficient memory space (Too deep nesting)

Reference:

Statements: `RETURN`

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 pro-gram 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.

Flow control statement

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`, `FUNC-TION ..END FUNCTION`, `IF ..THEN ..ELSE ..END IF`, `SELECT ..CASEEND SELECT`, `SUB ..END SUB`, and `WHILE ..WEND`), you can nest them to a maxi-mum 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, ONGOSUB, ON ..GOTO, SELECT ..CASE ..END SELECT, and WHILE ..WEND

I/O statement

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 or BS key 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 and echo back on the LCD, 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 (single-byte ANK mode or two-byte Kanji mode), the screen font size (standard-size or small-size), and the character enlargement attribute (regular-size, double-width, double-height, or quadruple-size). For details, refer to Chapter 7, Subsection 7.1.3.

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$` and `INPUT$`

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 *variables* 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 *variables*.

  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.

TIP

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.

- 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$`

KEY

Assigns a string or a control code to a function key; also defines a function key as a backlight function on/off key. This statement also defines a magic key as a 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 a backlight function on/off key):

`KEY` *`backlightkeynumber,onduration`*

Syntax 3 (Defining a magic key as a trigger switch, shift key, or battery voltage display key):

`KEY magickeynumber, "TRG"` (Trigger switch)
`KEY magickeynumber, "SFT"` (Shift key)
`KEY magickeynumber, "BAT"` (Battery voltage display key)

Parameter:

`keynumber`
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 single control code.

`backlightkeynumber`
A numeric expression which returns a value from 0 to 38.

`onduration`
Keyword BL and a string expression which returns a value from 0 to 255. (BL0 to BL255)

`magickeynumber`
30, 31, 35, or 36

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.")
- 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 a backlight function on/off key

KEY in syntax 2 defines a function key specified by *backlightkeynumber* as a backlight function on/off key and sets the length of backlight ON-time specified by *onduration*. (Refer to Chapter 13, "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.
- Specifying a zero (0) or 32 to *backlightkeynumber* defines the combination of the shift key and M4 key (key number 36) or the M4 key as a backlight function on/off key, respectively.

• Pressing the M1 key (key number 30) while holding down the shift key functions as a 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 a 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 a backlight function on/off key as shown below, only the function key numbered 8 operates as a 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 a trigger switch, shift key, or battery voltage display key

• `KEY` in syntax 3 defines a magic key as a trigger switch, shift key, or battery voltage display key, as well as assigning string data.

```
KEY 30,"TRG"            'M1 key as a trigger switch
KEY 31,"SFT"            'M2 key as a shift key
KEY 35,"BAT"            'M3 key as a battery voltage display 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 a 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 (combination of M1 key and shift key) will be restored.

```
KEY 3,"BL100"
KEY 3,"a"
```

The description below defines the M1 key as a trigger switch. The default battery voltage display key (combination of 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`

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 (which should be executed 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 a backlight function on/off key, trigger switch, shift key, or software keyboard 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.
- Specifying 32 to `keynumber` will be ignored.

■`KEY OFF`

`KEY OFF` disables k*`eystroke`* trapping for a function key specified by *`keynumber`*.

- 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 (*`key-number`*).

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 "[*drive-name*:]*filename*".

- [*drivename*:] is used in conventional BHT series. In the BHT-100 series, it is merely for the compatibility with their specifications. The *drivename* may be A: or B:, but it 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`

Kanji Pattern LOAD I/O statement

KPLOAD

Loads a user-defined Kanji font in the two-byte Kanji mode.
This statement also loads a user-defined cursor.

Syntax:

Syntax 1 (Loading a user-defined Kanji font):

`KPLOAD` *`kanjicode, fontarrayname`*

Syntax 2 (Loading a user-defined cursor):

`KPLOAD` *`kanjicode, cursorarrayname`*

Parameter:

`kanjicode`

- For a user-defined Kanji font

  A numeric expression which returns a value from EBC0h to EBFCh, 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; doing so will result in a syntax 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.
- Also when the small-size font or double-width is specified, user-defined fonts loaded by

  the `APLOAD` will be effective. The system will enlarge the dot pattern of each loaded font

  in small-size or double-width.

  For font patterns specified the small-size font or double-width, refer to Chapter 7,

  Subsection 7.1.3, "Dot Patterns of Fonts" and Subsection 7.1.5, "Displaying User-defined Characters."

■Loading a user-defined cursor

`KPLOAD` loads a user-defined cursor data defined by *cursorarrayname* to the user font area specified by *kanjicode*.

- To display a user-defined cursor loaded by the `KPLOAD`, you set 255 to *cursorswitch* in the `LOCATE` statement in the two-byte Kanji 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 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.

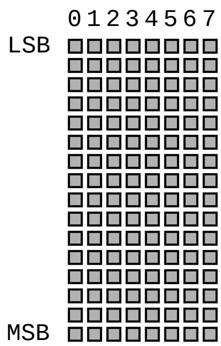
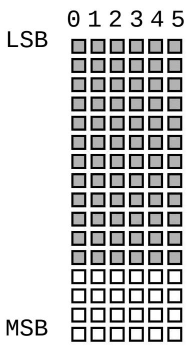
- An array integer variable--a work array, register array, or common array--for *`cursorarrayname`* should be declared by the `DIM`, `DEFREG`, or `COMMON` statement, respectively.

```
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.

<table>
<tr><th>Font size</th><th>Cursor size (W x H)</th><th>No. of elements</th></tr>
<tr><td>Standard-size</td><td>8 x 16 dots

</td><td>8</td></tr>
<tr><td>Small-size</td><td>6 x 12 dots

</td><td>6</td></tr>
</table>

- If the double-width is specified, then user-defined cursors loaded by the `KPLOAD` will display in double-width, respectively. For details, refer to Chapter 7, Subsection 7.1.3 "Dot Patterns of Fonts."

Syntax errors:

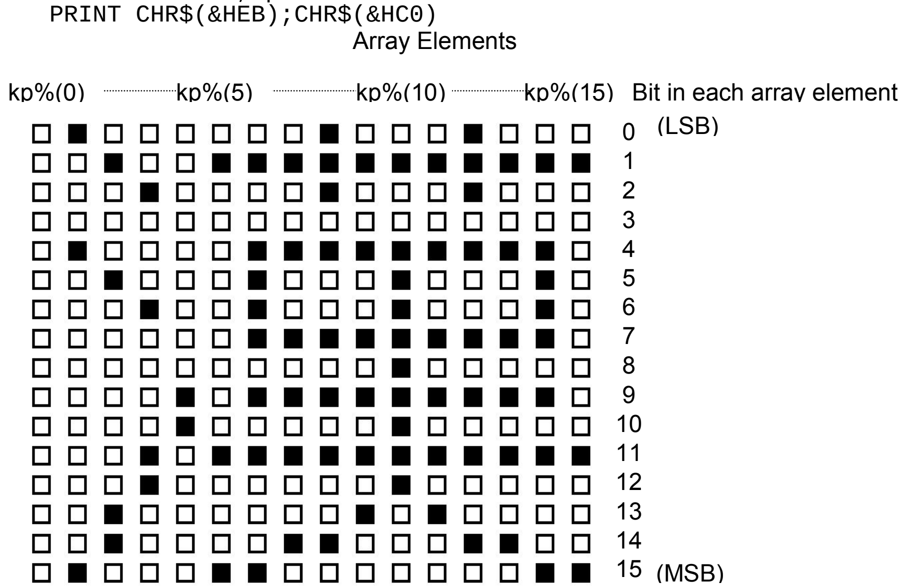
Error code and message	Meaning
`error 71: Syntax error`	• No *`fontarrayname`* or *`cursorarrayname`* is defined. • *`fontarrayname`* or *`cursorarrayname`* has an array string variable. • *`fontarrayname`* or *`cursorarrayname`* includes parentheses ( ). • *`fontarrayname`* or *`cursorarrayname`* includes 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)
```



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

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 (single-byte ANK mode or two-byte Kanji mode), the screen font size (standard-size or small-size), and the character enlargement attribute (regular-size, double-width, double-height, or quadruple-size). For details, refer to Chapter 7, Subsection 7.1.3.

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.

TIP

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 LINE 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.

Screen mode	Screen font	*column*	*row*	*cursorswitch*
Single-byte ANK Mode	Standard-size font	1 to 22	1 to 8	0 to 2, and 255
	Small-size font	1 to22	1 to 10	
Two-byte Kanji Mode	Standard-size font	1 to 17	1 to 7	0 to 2, and 255
	Small-size font	1 to 22	1 to 9	0 to 2, and 255

Description:

`LOCATE` moves the cursor to a position specified by *`column`* number and *`row`* number as coordinates on the LCD. It also changes the cursor shape as specified by *`cursorswitch`*.

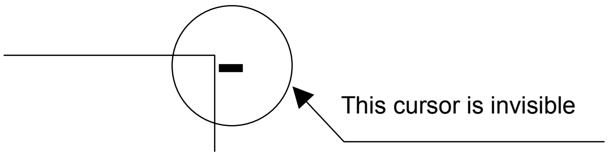
- The cursor location in the upper left corner of the LCD is 1, 1 which is the default.
- *`cursorswitch`* specifies the cursor shape as listed below.

`cursorswitch` value	Cursor shape
0	Invisible
1	Underline cursor (default)
2	Full block cursor
255	User-defined cursor

- 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.

- The entry ranges of the column and row are the same in the regular-size, double-width.
- Any parameter value outside its range will result in a run-time error.
- Specification of the maximum value to *column* moves the cursor off the screen and out of sight.

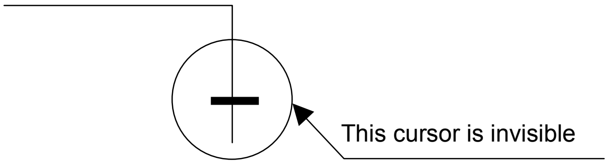
  Example: `SCREEN 0,0          'Regular size in ANK mode`
  `LOCATE 22`



  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.

- If the double-width or quadruple-size is specified, specification of the (maximum value - 1) to *column* moves the cursor off the screen and out of sight.

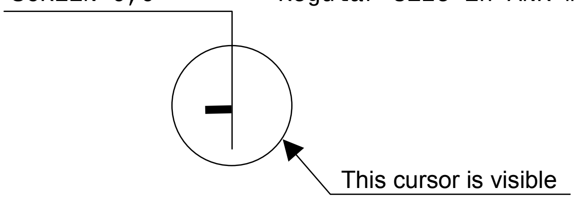
  Example: `SCREEN 0,2          'Double-width in ANK mode`
  `LOCATE 21`



  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.

  Switching to the regular-size will make the cursor visible as shown below.

  `SCREEN 0,0          'Regular size in ANK mode`



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 immediately 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, 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

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 rou-tine 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.
- 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 data 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
  ```
- If you set B: to [*drivename*], the specified file will be opened as a read-only file; if you set "A:" or omit [*drivename*], it 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), then the *filelength* will be ignored.
- Specifying only *filelength* does not allocate memory. 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. This statement also activates or deactivates the indicator LED and the beeper (vibrator) individually.

Syntax:

```
OPEN "BAR:[readmode][beepercontrol][LEDcontrol]" AS [#]filenumber
CODE readcode
[,readcode...]
```

Parameter:

readmode
A string expression.

beepercontrol
A string expression. Specification of B activates the beeper (vibrator).

(Default: Deactivated)

LEDcontrol
A string expression. Specification of L deactivates the green indicator 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 device (BHT) according to *filenumber*.

If the bar code device file has been opened with the `OPEN "BAR:"` statement, pressing the trigger switch [*1] turns on the illumination LED [*2] and makes the BHT ready to scan a bar code.

- If the BHT reads a bar code successfully, the indicator LED for reading confirmation will illuminate in green. Specification to *LEDcontrol* may activate or deactivate the indicator LED. Specification to *beepercontrol* may activate or deactivate the beeper and vibrator function.
- A bar code read will be decoded and then transferred to the barcode buffer.

[*1] The trigger switch function is assigned to the magic keys.
[*2] The illumination LED may not come on where the environment is bright enough for the BHT to scan.

- 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 cannot open the bar code device file and the IrDA 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 `readcodes` can be specified.
- If you specify more than one condition for a same bar code type with `readcode`(s) ("I" in the example below), all of those conditions are valid. The `sam-ple` 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"
```

- If you specify more than one `readcode` including "I" (ITF), then ITF codes less than 4 digits cannot be read unless numbers of digits are specified.

■*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 [*2] lights and the BHT can read a bar code.

If the bar code device file becomes closed when the trigger switch [*1] is helddown, the illumination LED 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 [*2] so that the BHT cannot read the next bar code.

[*1] The trigger switch function is assigned to the magic keys.
[*2] The illumination LED may not come on where the environment is bright enough for the BHT to scan.

□Auto-off mode (F)

```
OPEN "BAR :F"AS #7 CODE "A"
```

If you press the trigger switch [*1] , the illumination LED [*2] comes on. When you release the switch or when the BHT completes bar code reading, then the illumination LED will go off. Holding down the trigger switch [*1] lights the illumination LED for a maximum of 5 seconds.

While the illumination LED 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 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.

Once a bar code is read successfully, pressing the trigger switch [*1] cannot turn on the illumination LED [*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 [*2] comes on. Even if you release the switch, the illumination LED [*2] remains on until the bar code device file becomes closed or you press that switch again. While the illumination LED [*2] is on, the BHT can read a bar code.

Pressing the trigger switch [*1] toggles the illumination LED [*2] on and off.

Once a bar code is read successfully, pressing the trigger switch [*1] turns on the illumination LED [*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.

□Continuous reading mode (C)

```
OPEN "BAR :C"AS #7 CODE "A"
```

Upon execution of the above statement, the BHT turns on the illumination LED [*2] and keeps it on until the bar code device file becomes closed, irrespec-tive of the trigger switch [*1].

While the illumination LED [*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.

[*1] The trigger switch function is assigned to the magic keys.
[*2] The illumination LED may not come on where the environment is bright enough for the BHT to scan.

- 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*

The `OPEN "BAR:"` statement can control the beeper and the indicator LED to activate or deactivate each of them when a bar code is read successfully. The BHT 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.
- Specifying B to *beepercontrol* allows you to choose beeping only, vibrating only, or beeping & vibrating by making setting on the adjustment screen of the LCD contrast, beeper, and vibrator 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 indicator LED when a bar code is read successfully:

```
OPEN "BAR :L"AS #7 CODE "A"
```

■***readcode***

The BHT supports seven types of bar codes--the universal product codes, Interleaved 2 of 5 (ITF), Standard 2 of 5 (STF), Codabar (NW-7), Code 39, Code 93, and Code 128. The BHT can read also EAN-128 if Code 128 is specified.

(For readable bar code types, refer to the BHT User's Manual.)

□Universal product codes (A)

Syntax:

```
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 type
A	EAN-13, UPC-A
B	EAN-8
C	UPC-E

If *code* is omitted, the default is all of the universal product codes.

1stchara and *2ndchara* are flag characters representing a country code and should be numerals from 0 to 9. 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 to read also supplemental codes.

```
OPEN "BAR :"AS #1 CODE "A :49S"
```

□Interleaved 2 of 5 (ITF) (I)

Syntax :

```
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 99 and satisfy the following conditions:

mini.no.digits •max.no.digits

If both of *mini.no.digits* and *max.no.digits* are omitted, then 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:

```
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 99 and satisfy the following condition:

mini.no.digits •max.no.digits

If both of *mini.no.digits* and *max.no.digits* are omitted, then the default reading range is from the minimum number of digits specified in Sys-tem 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:

```
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 99, 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, then the default reading range is 1 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-43. The check digit is included in the number of digits.

```
OPEN "BAR:"AS #1 CODE "M:8-12C"
```

☐Code 93 （L）

Syntax:

```
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 99, 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, then the default reading range is 1 to 99 digits. 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 :

```
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 99, 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, then the default reading range is 1 to 99 digits. 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 digits 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. The beeper sounds only if it is activated.

FNC characters will be handled as follows:

(1) FNC1

The BHT 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.

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 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

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)

Syntax:

```
H[:[mini.no.digits[-max.no.digits]][CD] [startstop]
  [,[mini.no.digits[-max.no.digits]][CD] [startstop]]
  [,[mini.no.digits[-max.no.digits]][CD] [startstop]]]
```

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 99, 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, then 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, only the number of digits specified by *mini.no.digits* can be read.

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.

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 IrDA interface of the communications device file concurrently.)

OPEN "COM: "

Opens a communications device file.

Syntax:

Syntax 1 (For the direct-connect interface):

```
OPEN "COMn:[baud][,[parity][,[charalength][,[stopbit][,[RS/CS]
[,[timeout]]]]]] "AS [#] filenumber
```

Syntax 2 (For the IrDA interface):

```
OPEN "COMn: [baud] "AS [#] filenumber
```

Parameter:

baud

For the IrDA 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.

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

■*COMn*

`COMn` is a communications device file name.

The BHT supports both the IrDA and direct-connect interfaces but cannot open them concurrently. If you attempt to open both interfaces concurrently, a runtime error will occur.

Interface	Communications device file name
IrDA interface	"COM1 :"
Direct-connect interface	"COM2 :"
Default interface [*1]	"COM :"

*1 The default interface refers to an interface which is selected on the SET COMMUNICATION menu. (For details, refer to the BHT User's Manual.)

`COM` may be in lowercase as shown below.

```
OPEN "com :"AS #8
```

The BHT cannot open the IrDA interface and the bar code device file concurrently. If you attempt to open them concurrently, a run-time error will occur.

The BHT cannot open the Bluetooth communications device file concurrently with the IrDA interface or direct-connect interface. If you attempt to open them concurrently, a run-time error will occur.

■*baud*

When the IrDA 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, 9600 (default), 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 IrDA interface is compliant with the IrDA physical layer (IrDA-SIR1.2), 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*	IrDA I/F	Direct-connect I/F
0 (default)	Ignored	
1	Ignored	
2	Ignored	High RD will be regarded as a high CS.
3	Ignored	Low RD will be regarded as high CS.
4	Ignored	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.

If you specify it for the IrDA interface, it will be ignored resulting in no run-time error.

RS/CS option is also applicable to Busy control when the direct-connect inter-face is used. 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
```

■**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 100ms.

Specification of zero (0) causes no timeout.

To make the direct-connect interface 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 IrDA interface does not support timeout. If specified, the *timeout* option will be ignored resulting in no 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 IrDA interface of the communications device file concurrently.) (You attempted to concurrently open the two types of communications device files -- IrDA interface and Bluetooth interface, or direct-connect interface and Bluetooth interface.)

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: `INP`

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 according to the current display mode and character attributes. You need to select the display mode with a `SCREEN` statement before execution of the `PRINT` statement.
- 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␣␣␣␣_
```

(␣ is a space.)

- 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 (,) pre-ceding 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

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 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

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 #, then it is padded with spaces and right-justified.

Statement example: `PRINT USING "#####";123`

Output:

```
␣␣123
```

(␣ is a space.)

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, then 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 "*`stringformat`*".

PRIVATE

Declares one or more work variables or register variables defined in a file, to be private (aslocal variables).

Syntax:

Syntax 1:

```
PRIVATE varname [,varname...]
```

Syntax 2:

```
PRIVATE DEFREG registerdefinition [,registerdefinition...]
```

Parameter:

```
varname
   numericvar [(subscript)]
   stringvar [(subscript)[[stringlength]]]
registerdefinition
   non-arraynumericvar [=numericconstant]
   arraynumericvar(subscript) [=numericinitialvaluedefinition]
   non-arraystringvar[[stringlength]][=stringconstant]
   arraystringvar(subscript)[[stringlength]][=stringinitialvaluedef
   inition]
   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 which indicates the number of characters.

Description:

PRIVATE defines variables declared by *varname* or *registerdefinition* as local variables which can 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`

File I/O statement

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, then 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, then 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.
- 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` or `GET` 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 data in drive B

Reference:

Statements: `GET` and `OPEN`

I/O statement

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 run-time 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
P 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:

Statement example: DATA and RESTORE

Declarative statement

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 fol-lowing 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 labeled 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, or variable name/reserved wordused)`	`REM` begins in the first column of a program line.

Reference:

Statements: `$INCLUDE`

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

Error control statement

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

Flow control statement

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`

SCREEN

Sets the display mode (screen mode, and font size) and character attributes (character enlargement and font reverse attributes).

Syntax:

Syntax 1:

`SCREEN` *`displaymode`*`[,`*`charaattribute`*`]`

Syntax 2:

`SCREEN ,`*`charaattribute`*

Parameter:

`displaymode` and *`charaattribute`*

A numeric expression which returns a value from 0 to 3.

Description:

`displaymode` in the `SCREEN` statement sets screen mode and font size as listed below.

Screen mode	`SCREEN` *`displaymode`*
ANK mode	`SCREEN 0`
Kanji mode	`SCREEN 1`

`charaattribute` sets the character enlargement, and font reverse attributes as listed below.

Character enlarge-ment attribute	Font reverse attribute	`SCREEN ,charaattribute`
Regular	Normal	`SCREEN ,0`
	Reversed (Highlighted)	`SCREEN ,1`
Double-width	Normal	`SCREEN ,2`
	Reversed (Highlighted)	`SCREEN ,3`

- At the start of program execution, the following settings apply:

Screen mode	ANK mode
Font size	Standard-size
Character enlargement attribute	Regular
Font reverse attribute	Normal

- If *displaymode* or *charaattribute* parameter is omitted, the associated parameter value will not change.
- In the two-byte Kanji mode, characters can be displayed in either the full-width (16 dots wide by 16 dots high) or the half-width (8 dots wide by 16 dots high). If a small-size font is selected, those character sizes will become 12 dots wide by 12 dots high or 6 dots wide by 12 dots high, respectively.
- You may switch the font size by using the OUT statement (port &h6080). Refer to Chapter 14, OUT and Appendix D, "I/O Ports."

Run-time errors:

Error code	Meaning
05h	Parameter out of the range

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, test1, and *test2*

A numeric or string expression.

Description:

This statement executes one of *statementblocks* depending upon the value of *conditionalexpression* according to the steps below.

(1) SELECT evaluates *conditionalexpression* and compares it with *tests* 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 *tests* 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 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.)

SUB…END SUB

Names and defines user-defined 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, then 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` statement is used.
- `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 *`realparameters`* should be equal to that of *`dummyparameters`*, 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 *`realparameters`* 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 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-defined 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
A=1:B=2
PRINT "TEST"
add(A,B)
   ¦
```

File 2

```
SUB add(X,Y)
PRINT X+Y
END SUB
```

```
TEST
3
```

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.")

`ANDbyte` is a bit pattern in which bits to be checked should be set to 1. *`XORbyte`* is a bit pattern in which the same bit positions as ones set to 1 in *`ANDbyte`* should be set to the values to be picked out.

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, then 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`

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` cannot be written on a same program line.
- If no `WEND` 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`

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 communi-cations protocol specified by "*protocolspec*." (For the BHT-protocol and BHT-Ir protocol, refer to the BHT User's Manual.)

■"[*drivename*:]*filename*"

filename is a data file name. For the format of data file names, refer to the `OPEN` statement.

- [*drivename*:] is used in conventional BHT series. In the BHT-8000 series, it is merely for the compatibility with their specifications. The *drivename* may be A: or B:, but it will be ignored.

■"*protocolspec*"

"*protocolspec*" parameter can specify the following protocol specifications:

Specifications	BHT-protocol	BHT-Ir protocol
Transmission direction	✓	✓
Serial number	✓	
Horizontal parity checking (BCC)	✓	
Transmission monitoring	✓	✓
Handling of trailing space codes in a data field during file transmission	✓	✓
Timeout length when a link will be established	✓	✓
Checking whether filenames are identical	✓	✓

- 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"`

"*`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 trailing space codes in a data field during file transmission

Parameter omitted (default)	Trims space codes.
`T` or `t`	Handles space codes as data.

Example: `XFILE "d2.dat","T"`

Each of 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

  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 maximum number of ENQ retries is 20 (when the BHT-protocol is used.)

- Checking whether filenames are identical

  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 by *`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 "TEST2.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.
- A data file to be transmitted should be closed beforehand.
- To transfer a file by using the BHT-Ir 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"
"ABCDEFGHIJKLMNOPQRSTUVXYZ1"

"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
`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.)
`48h`	Device timeout
`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:"`

File I/O statement

$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.

If this statement is placed at the beginning of source programs, then same user-defined functions or subroutines may be shared by those source programs.

- *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 will not be 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 will not be 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 will be 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, then 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.

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 from 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, 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)

CDtype may be in lowercase.

- If *barcodedata* contains a character(s) out of the specification of the bar code type specified by *CDtype*, then 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 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 Codabar (NW-7), so CD.Data$ will become a null string.

Sample coding 1: `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 digits	EAN-13 or UPC-A
8 digits	EAN-8
7 digits	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 which returns any of `A, D, E, F, G, I, J, N, S,` and `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

- 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 SET DISPLAY menu in System Mode, respectively.

- After setting a national character set, you may display national characters assigned to 32 (20h) to 127 (7Fh), on the LCD.
- "*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 display mode specified by a SCREEN statement.

Screen mode	Font size	Row number
Single-byte ANK mode	Standard-size Small-size	1 to 8 1 to 10
Two-byte Kanji mode	Standard-size Small-size	1 to 7 1 to 9

- 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.
00 to 99 of *yy* is equal to 2000 to 2099.

■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.
  00 to 99 of *yy* is equal to 2000 to 2099.

- 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.
- If converted from decimals to hexadecimals with the HEX$ function, addresses which the ERL returns correspond to ones that are outputted to the left end of the address-source list in hexadecimal (which may be printed out if a +L option is specified in compilation).
- 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$

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.
- If converted from decimals to hexadecimals with the HEX$ function, codes which the ERR returns correspond to ones that are listed in Appendix A1, "Run-time Errors."
- 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$

End of TeXt I/O function

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 a data text in the BHT-protocol when the 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$` and `STX$`

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

- 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.)

HEXadecimal String function

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.

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$

INPort data I/O function

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$

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$` does not echo back read characters on the LCD screen.
- The cursor shape (invisible, underlined, or full block) depends upon the specification selected by the `LOCATE` statement.
- The cursor size depends upon the screen mode (single-byte ANK mode or two-byte Kanji mode), the screen font size (standard-size or small-size), and the character enlargement attribute (regular-size or double-width). For details about the cursor, refer to Chapter 7, Subsection 7.1.3.
- If any key previously specified for keystroke trapping is pressed during execution of the `INPUT$`, then 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	code Meaning
`05h`	Parameter out of the range (*startposition* is out of the range.)

Reference:

Functions: `LEN`

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(-0.5)
PRINT INT(-0.2)
```

```
-2
-1
```

LEFT String function

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
```

LOcation Counter of file File I/O function

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 99 bytes for bar codes.

- 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`

Location Of File | File I/O function

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	Error code
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 the bar code type and the number of digits of a read bar code.

Syntax:

```
MARK$
```

Description:

`MARK$` returns a 3-byte string which consists of the first one byte representing the bar code type and the remaining two bytes indicating the number of digits of a read bar code.

- The first one byte of a returned value contains one of the following letters representing code types:

Code type	First one byte of a returned value
EAN-13 , 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.

MIDdle String function

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$`, `LEN`, and `RIGHT$`

POSition | I/O function

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 display mode selected by a SCREEN statement, as an integer.

Screen mode	Font size	Column number
Single-byte ANK mode	Standard-size	1 to 22
	Small-size	1 to 22
Two-byte Kanji mode	Standard-size	1 to 17
	Small-size	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.
- The range of the column numbers does not differ between the regular-size 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$

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 File I/O function

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 record number 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 a record number 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 (record #1) 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`

Start Of Heading | I/O function

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$` and `STX$`

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, then STR$ automatically adds a leading space (meaning + sign) 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

Start of TeXt I/O function

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$

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 wakeup time):

```
TIME$
```

Syntax 2 (Setting the current system time or 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

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

`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$`

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.)

VALue String function

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-8000 series supports the following extended functions which can be invoked by the `CALL` statement.

Extended functions	Used to:	Remarks
`SYSTEM.FN3`	Read or write system settings from/to the memory.	
`SYSMDFY.FN3`	Reconfigure BHT system or get/set system reconfig file information.	
`CRC.FN3`	Calculate a CRC.	
`SOCKET.FN3`	Implement a subset of the TCP/IP socket application program interface (API). (For details, refer to Chapter 17.)	(Integrated in models equipped with the Bluetooth device)
`FTP.FN3`	Implement FTP client services for file transfer to/from FTP servers. (For details, refer to Chapter 17.)	(Integrated in models equipped with the Bluetooth device)
`BT.FN3`	Read or write Bluetooth parameters and control operation. (For details, refer to Chapter 18.)	(Integrated in models equipped with the Bluetooth device)

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:
.fcSysIGet	1	Read numeric data from System Mode settings
.fcSysISet	2	Write numeric data to System Mode settings
.fcSysSGet	3	Read string data from System Mode settings
.fcSysSSet	4	Write string data to System Mode settings
.fcFontInf	5	Get font information

16.2.2 Detailed Function Specifications

.fcSysIGet(=1) Read numeric data from System Mode settings

Syntax: `CALL "SYSTEM.FN3" .fcSysIGet 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
.sySFMode	1	Shift key mode	R/W	.sySFNlock	0	Nonlock	0
				.sySF1time	1	Onetime	
.syM1key	2	Assignment to M1 key	R/W	.syMkyNone	0	None	0
				.syMkyEnt	1	Enter key	
				.syMkyTrg	2	Trigger switch	
				.syMkySF	3	Shift key	
				.syMkyBL	4	Backlight on/off function key	
				-	5	(Reserved for system)	
.syM2key	3	Assignment to M2 key	R/W	Same as above			0
.syM3key	4	Assignment to M3 key	R/W	Same as above			2
.syM4key	5	Assignment to M4 key	R/W	Same as above			2
.syBarInvt	6	Black-and-white inverted label reading function	R/W	.syInvtOff	0	OFF	0
				.syInvtOn	1	ON	
-	7	(Reserved for system)	-				
.syDecdLvl	8	Decode level	R/W	1 to 9			4
.syITFMin	9	Minimum number of digits to be read for ITF	R/W	2 to 20			4
.sySTFMin	10	Minimum number of digits to be read for STF	R/W	1 to 20			3
.syNW7Min	11	Minimum number of digits to be read for Codabar	R/W	3 to 20			4

*1 R/W: Read and write possible

Item number (PARA%)		System menu item	Attribute *1	DATA%, numeric data of the system menu item			Initial value
.syCmifApl	12	Default interface to be used for user programs	R/W	.syCmifOpt	0	IrDA interface	0
				.syCmifCon	1	Direct-connect interface	
.syCmifSys	13	Default interface to be used for System Mode	R/W	.syCmifOpt	0	IrDA interface	0
				.syCmifCon	1	Direct-connect interface	
.syTrSpdOp	14	Transmission speed for IrDA interface	R/W	.syOp24	0	2400bps	5
				.syOp96	1	9600bps	
				.syOp192	2	19200bps	
				.syOp384	3	38400bps	
				.syOp576	4	57600bps	
				.syOp1152	5	115200bps	
-	15	(Reserved for system)	-				
-	16	(Reserved for system)	-				
-	17	(Reserved for system)	-				
.syTrSpdCn	18	Transmission speed for direct-connect interface	R/W	.syCn3	0	300bps	9
				.syCn6	1	600bps	
				.syCn12	2	1200bps	
				.syCn24	3	2400bps	
				.syCn48	4	4800bps	
				.syCn96	5	9600bps	
				.syCn192	6	19200bps	
				.syCn384	7	38400bps	
				.syCn576	8	57600bps	
				.syCn1152	9	115200bps	
.syVPrtyCn	19	Vertical parity for direct-connect interface	R/W	.syVPrtyN	0	None	0
				.syVPrtyO	1	Odd	
				.syVPrtyE	2	Even	
.syDatLnCn	20	Character length for direct-connect interface	R/W	syDatLen7	0	7 bits	1
				.syDatLen8	1	8 bits	
.syStpLnCn	21	Stop bit length for direct-connect inter face	R/W	.syStpLen1	0	1 bit	0
				.syStpLen2	1	2 bits	
.sySNoOp	22	Serial numbers for IrDA interface	R/W	.sySNoOff	0	No numbers (OFF)	1
				.sySNoOn	1	Add numbers (ON)	

*1 R/W: Read and write possible

Item number (PARA%)		System menu item	Attribute *1	DATA%, numeric data of the system menu item			Initial value
.syHPrtyOp	23	Horizontal parity for IrDA interface	R/W	.syHPtyOff	0	No parity (OFF)	1
				.syHPtyOn	1	Add (ON)	
.syLnkTmOp	24	Timeout for data link establishment for IrDA interface	R/W	.syLnkT0	0	No timeout	1
				.syLnkT30	1	30 sec	
				.syLnkT60	2	60 sec	
				.syLnkT90	3	90 sec	
				.syLnkT120	4	120 sec	
.syFldSpOp	25	Trailing spaces in a data field for IrDA interface	R/W	.sySpIgnr	0	Ignore (Trim)	0
				.sySpData	1	Handle as data	
.sySNoCn	26	Serial numbers for direct-connect interface	R/W	.sySNoOff	0	No numbers (OFF)	1
				.sySNoOn	1	Add numbers (ON)	
.syHPrtyCn	27	Horizontal parity for direct-connect interface	R/W	.syHPtyOff	0	No parity (OFF)	1
				.syHPtyOn	1	Add (ON)	
.syLnkTmCn	28	Timeout for data link establishment for direct-connect interface	R/W	.syLnkT0	0	No timeout	1
				.syLnkT30	1	30 sec	
				.syLnkT60	2	60 sec	
				.syLnkT90	3	90 sec	
				.syLnkT120	4	120 sec	
.syFldSpCn	29	Trailing spaces in a data field for direct-connect interface	R/W	.sySpIgnr	0	Ignore (Trim)	0
				.sySpData	1	Handle as data	
.syCmPrtcl	30	Communications protocol type	R/W	.syCPBHT	0	BHT protocol	2
				.syCPBHTIr	2	BHT-Ir protocol	
.syResm	31	Resume function	R/W	.syResmOff	0	OFF	1*2
				.syResmOn	1	ON	
-	32	(Reserved for system)	-				
-	33	(Reserved for system)	-				
-	34	(Reserved for system)	-				
.syRamSize	35	RAM size	RO	512/1024/2048 (kilobytes)			*3
.syRomSize	36	ROM size	RO	2048/4096/8192 (kilobytes)			*3
.syClstSize	37	Cluster size	RO	4096 (bytes)			

*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.

.fcSysISet(=2) **Write numeric data to System Mode settings**

Syntax: `CALL "SYSTEM.FN3" .fcSysISet PARA%,DATA%`

Description: This function 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.

.fcSysSGet(=3) **Read string data from System Mode settings**

Syntax: `CALL "SYSTEM.FN3" .fcSysSGet PARA%,DATA$`

Description: This function 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
.syVersion	1	System version	RO	"X.XX" fixed to 4 characters
—	2	(Reserved for system)	-	
.syModel	3	Model name	RO	Max. of 8 characters (e.g., "BHT75")
.syPrdctNo	4	Product number assigned to the BHT	RO	Fixed to 16 characters (e.g., "496310….")
.syBHTSNo	5	Serial number assigned to the BHT	R/W	Fixed to 6 characters
.syExePrg	6	Execution program	R/W	R/W Filename.xxx (Filename followed by period and extension) If not selected, a null string

.fcSysSSet(=4) Write string data to System Mode settings

Syntax: `CALL "SYSTEM.FN3" .fcSysSSet PARA%,DATA$`

Description: This function 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.

.fcFontInf(=5) Get font information

Syntax: `CALL "SYSTEM.FN3" .fcFontInf N.FONT%,VERSION$()`

Description: This function 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, then the `SYSTEM.FN3` returns the sets of the font information by the number of elements.

16.3 Controlling system files (`SYSMDFY.FN3`)

16.3.1 Function Number List of `SYSMDFY.FN3`

The SYSMDFY.FN3 may reconfigure the BHT system , as well as getting/setting system reconfig file information, depending upon the function number specified, as listed below.

Function number		Used to:
.fcMdBVGet	1	Get version of BHT system reconfig file
.fcMdBDo	2	Reconfigure BHT system
.fcMdBNGet	3	Get filename of BHT system reconfig file
.fcMdBNSet	4	Set filename of BHT system reconfig file

16.3.2 Detailed Function Specifications

.fcMdBVGet(=1) Get version of BHT system reconfig file

Syntax: `CALL "SYSMDFY.FN3" .fcMdBVGet FILE$, VERSION$`

Description: This function returns the version (VERSION$) of the BHT system reconfig file specified by FILE$.

Parameter: `FILE$` Filename

Returned value: `VERSION$` Version, 4 characters fixed

Run-time errors:

Error code	Meaning
32h	File type mismatch

.fcMdBDo(=2) **Reconfigure BHT system**

Syntax: `CALL "SYSMDFY.FN3" .fcMdBDo FILE$, OPT%`

Description: This function automatically reconfigures the BHT system by using the BHT system reconfig file specified by `FILE$`.

Parameter: `FILE$` Filename

`OPT%` Task after system reconfiguration

.smPwOff	0	Power off
.smReset	1	Reset the system software

Returned value: (None)

Run-time errors:

Error code	Meaning
32h	File type mismatch

.fcMdBNGet(=3) **Get filename of BHT system reconfig file**

Syntax: `CALL "SYSMDFY.FN3" .fcMdBNGet FILE$`

Description: This function returns the filename of the BHT system reconfig file to be used in System Mode, in `FILE$`.

Parameter: `FILE$` Filename consisting of drive name and file name, max. 14 characters (No drive name might be returned.)

Returned value: (None)

.fcMdBNSet(=4) **Set filename of BHT system reconfig file**

Syntax: `CALL "SYSMDFY.FN3" .fcMdBNSet FILE$`

Description: This function sets the filename (specified by `FILE$`) of the BHT system reconfig file to be used in System Mode.

Parameter: `FILE$` Filename consisting of drive name and file name, max. 14 characters (Drive name omissible)

Returned value: (None)

16.4 Calculating a CRC (CRC.FN3)

16.4.1 Function Number List of CRC.FN3

The CRC.FN3 may calculate a CRC depending upon the function number specified, as listed below.

Function number		Used to:
.fcCcitt	2	Calculate a CRC-CCITT.
.fcCrc16	3	Calculate a CRC-16.

16.4.2 Detailed Function Specifications

.fcCcitt(=2) Calculate a CRC-CCITT

Syntax: `CALL "CRC.FN3" .fcCcitt STRING1$, [ STRING2$, [...,]] CRC$`

Description: This function calculates a CRC of character strings specified by STRING1$, STRING2$, ...STRING8$ and returns the calculation result in CRC$.
Up to eight character strings may be specified by assigning them to non-array string variables.

Parameter: STRING1$, STRING2$, ...STRING8$
Non-array string variables that are operands of CRC gen-eration

Returned value: CRC$ Non-array string variable that stores the calculation result (2 characters, fixed length. In the 1st character position is the upper byte of the calculation result.)

.fcCrc16(=3) Calculate a CRC-16

Syntax: `CALL "CRC.FN3" .fcCrc16 STRING1$, [ STRING2$, [...,]] CRC$`

Description: This function calculates a CRC of character strings specified by STRING1$, STRING2$, ...STRING8$ and returns the calculation result in CRC$.
Up to eight character strings may be specified by assigning them to non-array string variables.

Parameter: STRING1$, STRING2$, ...STRING8$
Non-array string variables that are operands of CRC gen-eration

Returned value: CRC$ Non-array string variable that stores the calculation result (2 characters, fixed length. In the 1st character position is the upper byte of the calculation result.)

Chapter 17
TCP/IP

(BHTs with Bluetooth communications device)

CONTENTS

17.1 Two Sides

17.1.1 BHT

The BHT equipped with a spread spectrum communications device or IrDA communications device 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 or IrDA communication system, respectively.

`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.

17.1.2 Hosts

`SOCKET.FN3` and `FTP.FN3` require a host machine with the equivalent TCP/IP functionality and running the appropriate server software.

17.2 Programming Procedure

17.2.1 Bluetooth Communication System

The following is the procedure for using TCP/IP over a Bluetooth communications device.

For programming details, refer to the sample source program separately provided.

[1] Open Bluetooth Communications Device

Connect to modem with Bluetooth device and establish data link using the modem.

For further details, refer to Section Chapter 18 "Bluetooth (BHTs with Bluetooth communications device)," Subsection 18.2.3.3.

[2] Configure TCP/IP System

To connect to the TCP/IP pathway, specify the following system settings by using the extension library SOCKET.FN3 in a user program:

- IP address
- Subnet mask
- Default gateway
- PPP authentication procedure
- User name for PPP authentication
- Password for PPP authentication

These settings will be used in [4 .]

For the details of the SOCKET.FN3, refer to Section 17.5 "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
ppp.auth%  = .soPPPPAP                             'PPP authentication procedure
ppp.usr$   = "USER"                                'User name for PPP
ppp.psw$   = "PASSWORD"                            'Password for PPP
para% = 1                                          'Specify IP address (#1)
call "socket.fn3" .fcTSysSet para%, my.addr$
para% = 2                                          'Specify subnet mask (#2)
call "socket.fn3" .fcTSysSet para%, subnetmask$
para% = 3                                          'Specify default gateway (#3)
call "socket.fn3" .fcTSysSet para%, gateway$
para% = 4                                          'Specify PPP authentication (#4)
call "socket.fn3" .fcTSysSet para%, ppp.auth%
para% = 5                                          'Specify User name for PPP (#5)
call "socket.fn3" .fcTSysSet para%, ppp.usr$
para% = 6                                          'Specify Password for PPP (#6)
call "socket.fn3" .fcTSysSet para%, ppp.psw$
```

[3] Declare TCP/IP Communications Pathway

Specify the following system settings by using the socket library (SOCKET.FN3):

- Communications device: Bluetooth communications device
- Link layer: PPP

For the setting procedure with the SOCKET.FN3, refer to Section 17.5 "Socket Library (SOCKET.FN3)."

Given below is a setting example using SOCKET.FN3:

```
iftype% = .soDvCOM4            'Specify Bluetooth communications device
layermode% = .soLyPPP          'Specify PPP as a link layer

call "socket.fn3" .fcTSetup iftype%, layermode%, interface%
                               'Specify communications pathway
                               '(SOCKET.FN3 function #40)
                               'Returns value in interface%
                               '(The returned value will be used in
                               '[4] and [6].)
```

[4] 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
- PPP authentication procedure
- User name for PPP authentication
- Password for PPP authentication

There are two ways to specify these parameters.

(a) Use the system settings with the extension library SOCKET.FN3. Refer to Section 17.5 "Socket Library (SOCKET.FN3)."

Given below is an example using SOCKET.FN3.

```
call "socket.fn3" .fcTCnnSys 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 17.5 "Socket Library (SOCKET.FN3)."

Given below is an example using SOCKET.FN3.

```
my.adr$   = "192.168.0.125"             'IP address of the BHT
subnet$   = "255.255.255.0"             'Subnet mask
gw$       = "0.0.0.0"                   'Default gateway
auth%     = .soPPPPAP                   'PPP authentication procedure
usr$      = "USER"                      'User name for PPP
psw$      = "PASSWORD"                  'Password for PPP
call "socket.fn3" .fcTCnnUsr interface%,my.adr$,subnet$,gw$,auth%,usr$,psw$
                                        'Connect to communications pathway
                                        'Use the returned value of [3] in
                                        'interface%.
```

[5] Transfer Data or File via Socket Interface

To transfer data via the socket interface, use the extension library SOCKET.FN3. Refer to 17.3, "Socket API" and Section 17.5 "Socket Library (SOCKET.FN3)."

To transfer file via the socket interface, refer to Section 17.4.3, "Using FTP Client."

[6] Disconnect TCP/IP Communications Pathway

Use the extension library SOCKET.FN3. Refer to Section 17.5 "Socket Library (SOCKET.FN3)."

Given below is an example using SOCKET.FN3.

```
call "socket.fn3" .fcTDiscnn interface%     'Disconnect TCP/IP communications
                                            'pathway (SOCKET.FN3 function #43)
                                            'Use the returned value of [3] in
                                            'interface%.
```

[7] Close Bluetooth Communications Device

Disconnect data link using a modem.

For further details, refer to Section Chapter 18 "Bluetooth (BHTs with Bluetooth communications device)," Subsection 18.2.3.3.

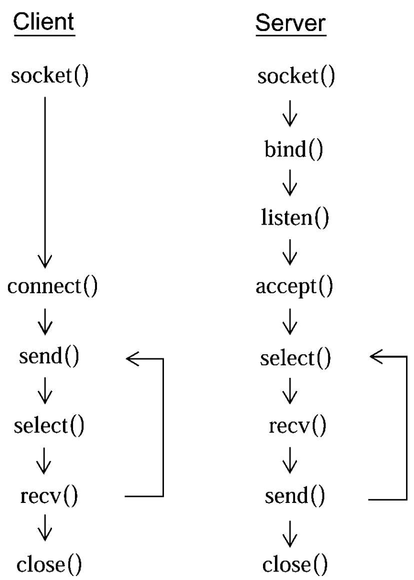
17.3 Socket API

17.3.1 Overview

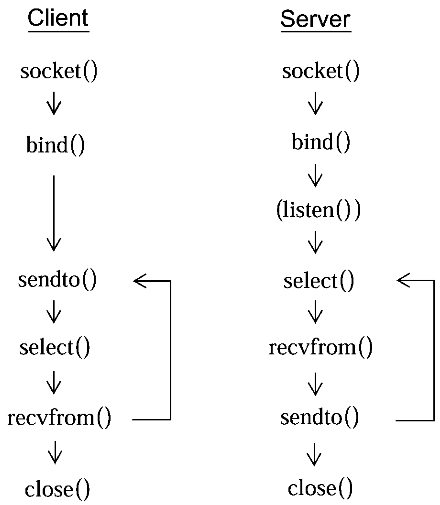
The `SOCKET.FN3` library implements a subset of the BSD4.4 socket application program interface (API).

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)



17.3.2 Programming Notes for Socket API

[1] Programming Notes for TCP

(a) Avoid retransmission control in application programs (recommended)

The TCP has flow control and retransmission control, so incorporating retransmission control into communication programs using the TCP socket may cause send data to be double sent or unintended data to be received.

When using the TCP socket, therefore, do not incorporate retransmission control in applications.

If an error occurs in TCP socket communication, close the socket once, then open it and start communication from the beginning again.

(b) Modify the status retaining period (recommended)

Socket API according to the TCP/IP is restricted by the following specifications. For the extended function `SOCKET.FN3` given below, refer to Section 17.5 "Socket Library (`SOCKET.FN3`)."

(1) After closed, the TCP socket will retain data for 60 seconds to keep the current status. For the 60 seconds, therefore, the socket cannot be used again.

(2) `SOCKET.FN3` function #26 may create a maximum of 64 sockets.

(3) The TCP/IP will function from when `SOCKET.FN3` function #41 or #42 connects the TCP/IP communications pathway until `SOCKET.FN3` function #43 disconnects it. Except for this period, timers used in the TCP/IP will stop.

In programming for TCP socket communication, if the period from connection to disconnection of the TCP/IP communications pathway is too short (approx. 1 second), then an error may occur. In the sample below, when the 65th socket is created, a run-time error (error code: &h218h) may occur indicating too many sockets created.

To avoid occurrence of run-time errors, set socket options (SOCKET.FN3 function #24) following TCP socket creation (SOCKET.FN3 function #26).

```
optname%=29                             'Set status retaining period after
option%=0                               'closing TCP socket to 0 second
                                        '(release immediately)
call "socket.fn3" .fcSSckOpt sockfd%, optname%, option
                                        'Set socket options
                                        'Use SOCKET.FN3 function #24.
```

[2] Programming Notes for 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.

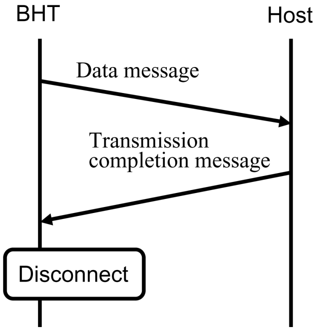
Given below are message transmission examples that support retransmission controls at each of the BHT and host.

■ BHT's retransmission control for a transmission error

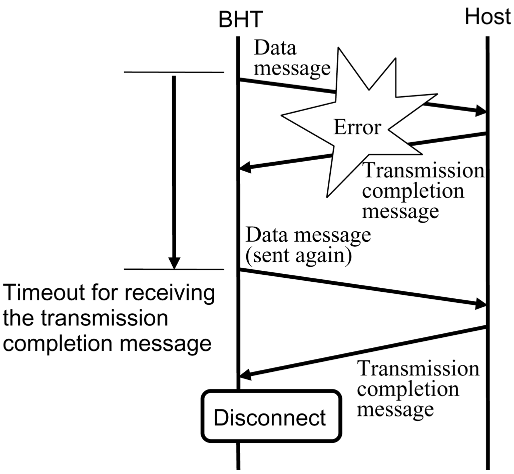
Assume that the BHT uses the protocol of receiving transmission completion message from the host after sending a message.

If the BHT times out for waiting a transmission completion message, it will transmit the unsent message again.

Normal end



Transmission error in a message sent from the BHT

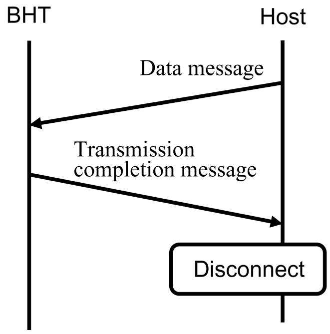


■ Host's retransmission control for a transmission error

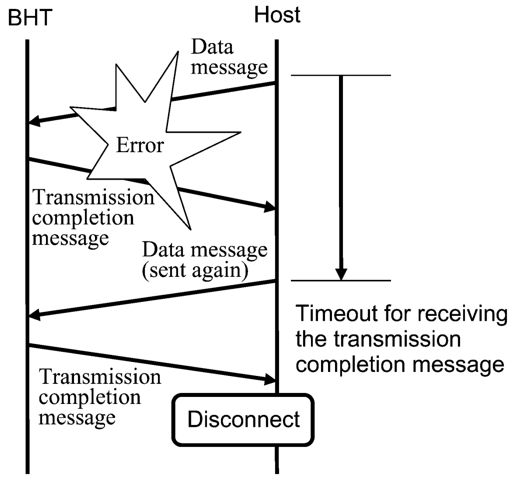
Assume that the host uses the protocol of receiving transmission completion message from the BHT after sending a message.

If the host times out for waiting a transmission completion message, it will transmit the unsent message again.

Normal end



Transmission error in a message sent from the host



[3]   Programming Notes for Socket API

If TCP/IP communication becomes no longer possible during data transmission, extended functions SOCKET.FN3 and FTP.FN3 will return any of the following run-time errors will be returned. For details about those extended functions, refer to Sections 17.5, "Socket Library (SOCKET.FN3)" and 17.6, "FTP Library (FTP.FN3):"

Run-time errors:

Error code	Meaning
105h	Power-off detected. (The BHT has been turned off during data transmission and then turned on. The communications device remains off.)
106h	An internal error has occurred in the TCP/IP module during data transmission.
108h	The memory for the TCP/IP module has became insufficient during data transmission.

■ Error recovery procedure from run-time errors 105h, 106h, and 108h

(1) Use the ON ERROR GOTO statement for error interrupt (In the error-handling routine, 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 communications device (This step is applicable to error 105h.)

Given below is a sample program for error occurrence. (This sample shows only the skeleton of communication program and requires modification in actual programming as necessary.)

(Example)

```
  STATUS% = 0
  ON ERROR GOTO TCP.ERR               'Prepare for error interrupt (To TCP.ERR
                                  'at the time of error occurrence)
DEV.OPEN:
  '<<<<< Open communications device processing (OPEN "COM1" / OPEN "COM3:") >>>>>
  STATUS% = 1

TCP.CONNECT:
  '<<<<< Connect to TCP/IP Communications pathway processing >>>>>
  '<<<<< (CALL "SOCKET.FN3" 41 / 42)                    >>>>>
  STATUS% = 2

TCP.SOCKET:
  '<<<<< Create socket processing (CALL "SOCKET.FN3 26) >>>>>
  STATUS% = 3

  '<<<<< Transfer data or file processing via socket interface >>>>>

  '<<<<< Close the socket processing (CALL "SOCKET.FN3 28) >>>>>
  STATUS% = 2

  '<<<<< Disconnect TCP/IP communications pathway processing >>>>>
  '<<<<< (CALL "SOCKET.FN3 43)                          >>>>>
  STATUS% = 1

  '<<<<< Close communications device processing (CLOSE) >>>>>
  STATUS% = 0
  ON ERROR GOTO 0

  RETURN

'*****************************************************
'  Error-handling routine processing
'*****************************************************
TCP.ERR:
  WERR = ERR
  RESUME ERRSUB

ERRSUB:
  ON ERROR GOTO ERRSUB2
```

```
  IF STATUS% > 2 THEN
     '<<<<< Close the socket processing (CALL "SOCKET.FN3 28) >>>>>
    IF (WERR<>&h105) AND (WERR<>&h106) AND (WERR<>&h108) THEN
      STATUS% = 2
      GOTO TCP.SOCKET
    ENDIF
  ENDIF

  IF STATUS% > 1 THEN
    '<<<<< Disconnect TCP/IP communications pathway processing >>>>>
    '<<<<< (CALL "SOCKET.FN3 43)                           >>>>>
    IF (WERR<>&h105) THEN
      STATUS% = 1
      GOTO TCP.CONNECT
    ENDIF
  ENDIF

  IF STATUS% > 0 THEN
   '<<<<< Close communications device processing (CLOSE) >>>>>
    STATUS% = 0
    GOTO DEV.OPEN
  ENDIF

  ON ERROR GOTO 0
  RETURN


ERRSUB2:

  RESUME NEXT
```

■ Note for run-time error 105h

Socket close processing (SOCKET.FN3, Function #28) following occurrence of run-time error 105h would not complete immediately. This is because a FIN packet will be transmitted repeatedly in the socket close processing until the communications device receives any response from the server independent of the power on/off state of the communications device.

The socket close processing period may be shortened by changing the retry count that determines the number of FIN packet retransmission times and is controlled by SOCKET.FN3, function #24, option #26.

(Example)

```
Sock.Err:                                'Socket error-handling routine
                                         'processing
  print "ERR:";hex$(err.code%)           'Display error code
  print "ERL:";hex$(err.line%)           'Display error line number
  if sock.stts%>=3 then                  'If OK until socket generation,
    optname%=26                          'set retry count
    option=0                             'No retry (transmit once)
    call "socket.fn3" 24 sockfd%,optname%,option
    call "socket.fn3" 28 sockfd%         'Close socket
  end if
    if sock.stts%>=2 then                'If OK until connection of TCP/IP
                                         'communications pathway,
    call "socket.fn3" 43 interface%      'Disconnect the pathway
  end if
  if sock.stts%>=1 then                  'If OK until opening the spread
                                         'spectrum communications device,
    close #hCom%                         'close the device
  end if
  goto main                              'To main program
```

17.4 FTP Client

17.4.1 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.

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.

17.4.2 File Formats

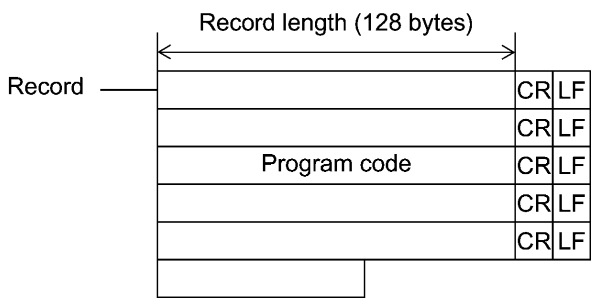
The FTP client classifies files into three types by their extensions: user programs (*.PD3), extension libraries (*.FN3 and *.EX3), and data files (other extensions).

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).

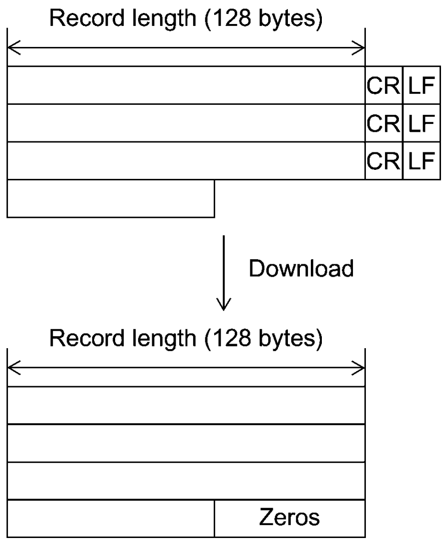
[1] User Programs (*.PD3)

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).

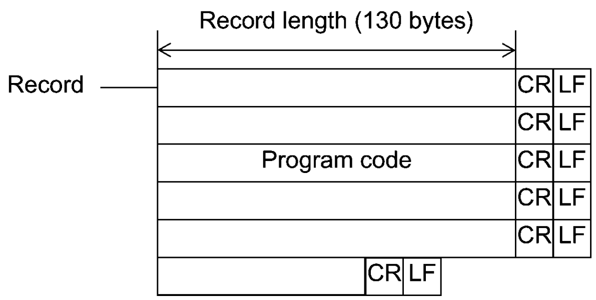


Aside: To conserve memory and boost performance, the BHT packs a pair of ASCII bytes into a single byte by converting each byte into a 4-bit hexadecimal number.

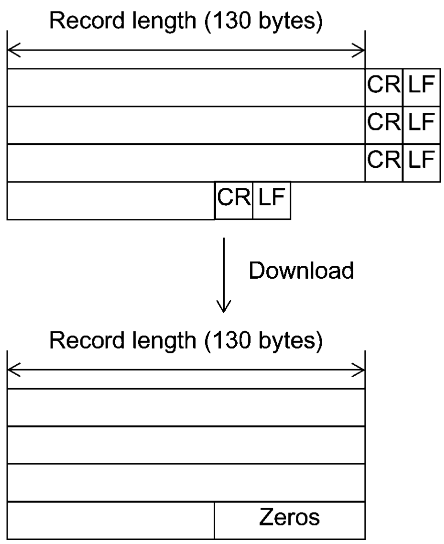
[2] Extension Libraries (*.FN3 and *.EX3)

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.)



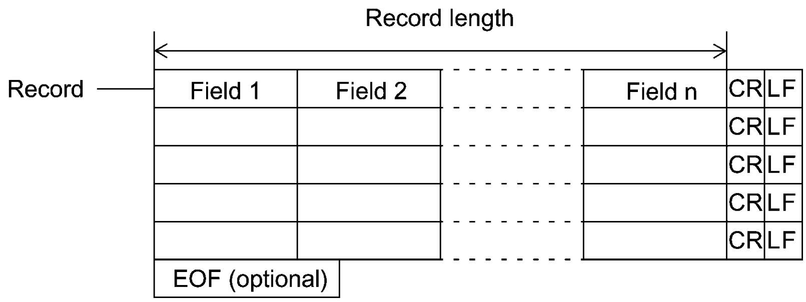
Aside: When downloading extension libraries, the BHT uses 128 bytes out of 130 bytes of record length (the remaining 2 bytes will be used for checking data). To conserve memory and boost performance, the BHT packs a pair of ASCII bytes into a single byte by converting each byte into a 4-bit hexadecimal 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.

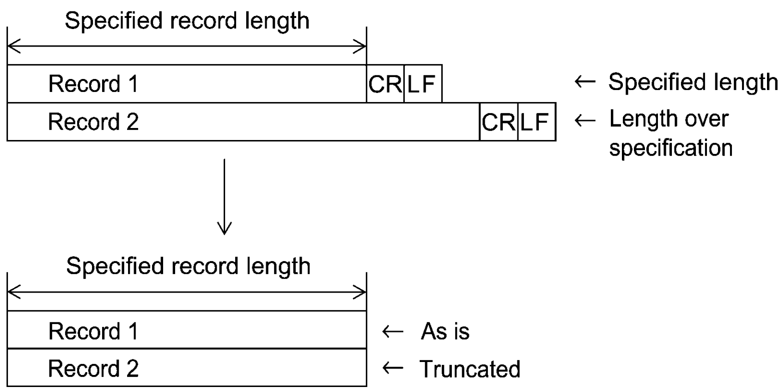
Data files are not limited to ASCII characters. They can use all bytes codes from 00h to FFh.



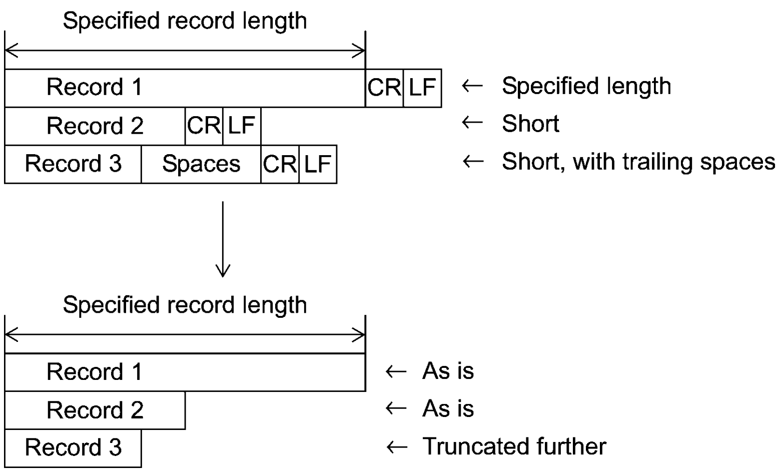
There can be 1 to 16 fields, each 1 to 254 bytes long. The sum of the field lengths and the number of fields, however, must not exceed 255.

If the actual record length is different from the specified record length

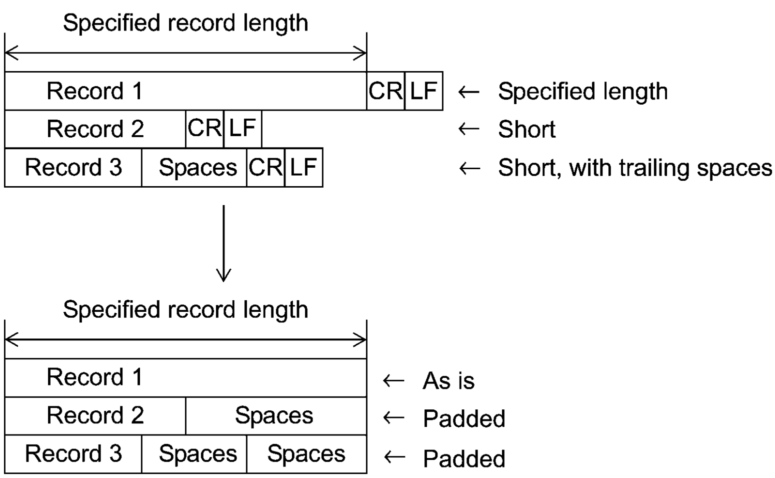
The FTP client discards any excess beyond the specified record length during downloads.



The treatment of short records is under application control. The default is to delete any trailing spaces (20h).

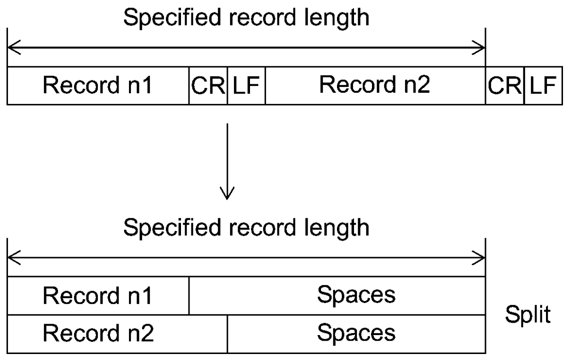


Alternatively, the FTP client can pad such short records to the specified record length with spaces (20h).

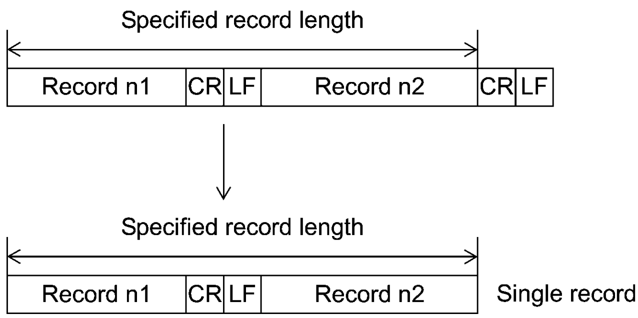


Line delimiters inside data records

The FTP client can send and receive all codes from 00h to FFh as described above. The treatment of line delimiters (CR-LF, CR, or LF) inside downloaded data records is under application control. The default 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.



17.4.3 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 extension library `FTP.FN3`.

(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 17.2, "Programming Procedure." The key step is to use the `FTP.FN3` 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

`FTP.FN3` provides functions #8 and #9 for reading and changing these settings. For further details on these two functions, Refer to their descriptions in Section 17.6, "FTP Library (`FTP.FN3`)," Subsection 17.6.2.

[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 BHT uses that larger block size.
- HFS = host file size

■ User Programs (*.PD3)

Determine MEM from HFS.

BFS = ROUND_UP (HFS ÷ (128 + LDS)) × 64

MEM = ROUND_UP (BFS ÷ 4096) × 4096

Example: File size of 12,345 bytes on operating system using CR-LF combination

BFS = ROUND_UP (12345 ÷ (128 + 2)) × 64 = ROUND_UP(94.96) × 64 = 6080

MEM = ROUND_UP (6080 ÷ 4096) × 4096 = ROUND_UP(1.48) × 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 ÷ (130 + LDS)) × 64

MEM = ROUND_UP (BFS ÷ 4096) × 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_DOWN (4096 ÷ 36) = ROUND_DOWN (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 spread spectrum communications device is open.

[5] FTP Transfers

The following is the basic procedure for transferring files with the `FTP.FN3` extended 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.

17.5 Socket Library (`SOCKET.FN3`)

17.5.1 Overview

■ String Variables

The following are the string variables used by this library together with their memory requirements.

Description	Variable name	Size in bytes
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	Variable 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		Used to:	Corresponding Socket API Function
.fcAccept	1*	—	accept()
.fcBind	2	Assign address to socket	bind()
.fcConnect	3	Connect socket	connect()
.fcGPName	4*	—	getpeername()
.fcGSName	5*	—	getsockname()
.fcGSckOpt	6	Get socket option	getsockopt()
.fcHToNL	7	Convert host long (4 bytes) to network byte order	htonl()
.fcHToNS	8	Convert host short (2 bytes) to network byte order	htons()
.fcINetAdr	9	Convert Internet address from dotted quad notation to 32-bit integer	inet_addr()
.fcListen	10*	—	listen()
.fcNToHL	11	Convert network long (4 bytes) to host byte order	ntohl()
.fcNToHS	12	Convert network short (2 bytes) to host byte order	ntohs()
.fcReadv	13*	—	readv()
.fcRecv	14	Receive data sent to the specified TCP socket	recv()
.fcRcvfrom	15	Receive data sent to the specified UDP socket	recvfrom()
.fcResvPrt	16*	—	rresvport()
.fcSelect	17	Monitor socket requests	select()
.fcFDZERO	18	Initialize socket identifier set	FD_ZERO macro
.fcFDSET	19	Add socket identifier to socket identifier set	FD_SET macro
.fcFDCLR	20	Delete socket identifier from socket identifier set	FD_CLR macro
.fcFDISSET	21	Get socket identifier status from socket identifier set	FD_ISSET macro
.fcSend	22	Send message to another TCP socket	send()
.fcSendto	23	Send message to another UDP socket	sendto()
.fcSSckOpt	24	Set socket options	setsockopt()
.fcShutdwn	25	Shut down socket	shutdown()

* Socket API function not supported by SOCKET.FN3 library.

Number		Used to:	Corresponding Socket API Function
.fcSocket	26	Create socket	socket()
.fcWritev	27*	—	writev()
.fcClose	28	Close socket	close()
.fcTSetup	40	Specify TCP/IP communications pathway	Unique to BHT
.fcTCnnSys	41	Connect TCP/IP communications pathway with system settings	Unique to BHT
.fcTCnnUsr	42	Connect TCP/IP communications pathway with user settings	Unique to BHT
.fcTDiscnn	43	Disconnect TCP/IP communications pathway	Unique to BHT
.fcTSysGet	44	Get TCP/IP system settings	Unique to BHT
.fcTSysSet	45	Set TCP/IP system settings	Unique to BHT
.fcTStsGet	46	Get TCP socket status	Unique to BHT

* Socket API function not supported by SOCKET.FN3 library.

17.5.2 Detailed Function Specifications

Function #2 .fcBind
Assign address to socket

Syntax: CALL "SOCKET.FN3" .fcBind *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	IP address
IPADDRESS$	Internet address in dotted quad notation

The protocol family (*FAMILY%*) must be 2, the value indicating the ARPA Internet protocols.

.soINet	2	ARPA Internet protocols

When specifying the value greater than 32767, describe in hexadecimal notation.

Example: PORT% = &h8000' Specify Port 32768

Return value: (None)

Run-time errors:

Error code	Meaning
105h	Power-off detected.
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 `.fcConnect`
Connect socket

Syntax: `CALL "SOCKET.FN3" .fcConnect` *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.

.soINet	2	ARPA Internet protocols

When specifying the value greater than 32767, describe in hexadecimal notation.

Example: PORT% = &h8000' Specify Port 32768

Return value: (None)

Run-time errors:

Error code	Meaning
105h	Power-off detected
106h	An internal error has occurred in the TCP/IP module during data transmission.
107h	The TCP/IP module has not been initiated.
108h	The memory for the TCP/IP module has became insufficient during data transmission.
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.
23Ah	The specified TCP socket has been closed.
23Ch	The connection attempt has timed out.
23Dh	Failed to connect
293h	The problem occurred on the communication pathway.
241h	There is no connection pathway to the host for TCP socket.

Function #6 `.fcGSckOpt`
Get socket option

Syntax: `CALL "SOCKET.FN3" .fcGSckOpt` *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%*)		
`.soKepAliv`	2	Keep-alive timer enable/disable	`.soDisable`	0	Disabled
			`.soEnable`	1	Enabled

Option Number (*OPTNAME%*)		Description	Values for Option (*OPTION*)
`.soSndBuff`	8	Transmit buffer size (byte)	1 to 8192
`.soRcvBuff`	9	Receive buffer size (byte)	1 to 8192
`.soMaxRT`	26	Retry count	0 to 32
`.soTIMEWAIT`	29	Status retaining period after closing TCP socket (seconds)	0 to 60
`.soRTODef`	30	Initial round trip time (ms)*	100 to 3000
`.soRTOMin`	31	Minimum round trip time (ms)*	100 to 1000
`.soRTOMax`	32	Maximum round trip time (ms)*	100 to 60000

* Shown in units of 100. (e.g., 1 = 100 ms).

Run-time errors:

Error code	Meaning
`105h`	Power-off detected.
`106h`	An internal error has occurred in the TCP/IP module during data transmission.
`107h`	The TCP/IP module has not been initiated.
`108h`	The memory for the TCP/IP module has became insufficient during data transmission.
`209h`	Socket identifier is invalid.
`216h`	A parameter is invalid.

Function #7 `.fcHToNL`

Convert host long (4 bytes) to network byte order

Syntax: `CALL "SOCKET.FN3" .fcHToNL` *`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 `.fcHToNS`

Convert host short (2 bytes) to network byte order

Syntax: `CALL "SOCKET.FN3" .fcHToNS` *`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 `.fcINetAdr`

Convert Internet address from dotted quad notation to 32-bit integer

Syntax: `CALL "SOCKET.FN3" .fcINetAdr` *`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. `fcNToHL`

Convert network long (4 bytes) to host byte order

Syntax: `CALL "SOCKET.FN3" .fcNToHL` *`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. `fcNToHS`

Convert network short (2 bytes) to host byte order

Syntax: `CALL "SOCKET.FN3" .fcNToHS` *`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 `.fcRecv`

Receive data sent to the specified TCP socket

Syntax: `CALL "SOCKET.FN3" .fcRecv` *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 non-array 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:

`.soRvNrm`	0	Normal
`.soRvOOB`	1	Out of band data
`.soRvPeek`	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:

`.soRvApend`	0	Append data to buffer (default if omitted)
`.soRvWrite`	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
106h	An internal error has occurred in the TCP/IP module during data transmission.
107h	The TCP/IP module has not been initiated.
108h	The memory for the TCP/IP module has became insufficient during data transmission.
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 disconnection.
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'	Second half is appended to first.
⋮	⋮	⋮	⋮	⋮	⋮		⋮	⋮	⋮	⋮	
Element 7	'6'	'7'	'8'	'9'	'0'	....	'0'	'1'	'2'	'3'	

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

	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	‘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 `.fcRcvfrom`

Receive data sent to the specified UDP socket

Syntax:

`CALL "SOCKET.FN3" .fcRcvfrom` *SOCKFD%, RECVBUFF$[()], RECVLEN%, RECVMODE%, FAMILY%, PORT%, address, RECVSIZE% [,RECVFLAG%]*

where *address* is *ADDRESS* or *IPADDRESS$*

Description:

This function receives data sent to the UDP socket specified by the socket identifier.

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
RECVFLAG%	Storage method (optional)

The receive buffer (*RECVBUFF$*) can be either a string non-array or string array variable. The maximum size for a string non-array is 255 bytes; for a string array, 4096.

The receive mode (*RECVMODE%*) must be one of the following values:

`.soRvNrm`	0	Normal
`.soRvPeek`	2	Peek at next message

The protocol family (*FAMILY%*) must be 2, the value indicating the ARPA Internet protocols.

`.soINet`	2	ARPA Internet protocols

The storage method (*RECVFLAG%*) is required for a string array buffer. It is ignored for a string non-array variable and new data will be written.

The storage method (*RECVFLAG%*) must be one of the following values:

`.soRvApend`	0	Append data to buffer (default if omitted)
`.soRvWrite`	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:

FAMILY%	Protocol family of sending station
PORT%	Port number of sending station
ADDRESS	Address of sending station
IPADDRESS$	Address of sending station in dotted quad notation
RECVSIZE%	Number of bytes received

Run-time errors:

Error code	Meaning
`105h`	Power-off detected
`106h`	An internal error has occurred in the TCP/IP module during data transmission.
`107h`	The TCP/IP module has not been initiated.
`108h`	The memory for the TCP/IP module has became insufficient during data transmission.
`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.
`240h`	No receiver found.

Function #17 `.fcSelect`

Monitor socket requests

Syntax: `CALL "SOCKET.FN3" .fcSelect` *MAXFD%, READFDSET$, WRITEFDSET$, 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:

`.soNoWait`	`-1`	No waiting period
`.soNotTOut`	`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
`106h`	An internal error has occurred in the TCP/IP module during data transmission.
`107h`	The TCP/IP module has not been initiated.
`108h`	The memory for the TCP/IP module has became insufficient during data transmission.
`209h`	Socket identifier is invalid.
`216h`	A parameter is invalid.

Function #18 . fcFDZERO
Initialize socket identifier set

Syntax: `CALL "SOCKET.FN3" .fcFDZERO` *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 . fcFDSET
Add socket identifier to socket identifier set

Syntax: `CALL "SOCKET.FN3" .fcFDSET` *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 . fcFDCLR
Delete socket identifier from socket identifier set

Syntax: `CALL "SOCKET.FN3" .fcFDCLR` *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 .fcFDISSET
Get socket identifier status from socket identifier set

Syntax: CALL "SOCKET.FN3" .fcFDISSET *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%*) must be one of the following values:

.soFDSet	0	No change
.soFDNoSet	1	Change in status

Function #22 .fcSend

Send message to another TCP socket

Syntax: `CALL "SOCKET.FN3" .fcSend` *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 sendto() 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 non-array 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:

.soSdNrm	0	Normal
.soSdOOB	1	Out of band data
.soSdDnRt	4	Bypass pathway control function

Return value: *SENDSIZE%* Number of bytes transmitted

Run-time errors:

Error code	Meaning
105h	Power-off detected
106h	An internal error has occurred in the TCP/IP module during data transmission.
107h	The TCP/IP module has not been initiated.
108h	The memory for the TCP/IP module has became insufficient during data transmission.
209h	Socket identifier is invalid.
216h	A parameter 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 `.fcSendto`

Send message to another UDP socket

Syntax: `CALL "SOCKET.FN3" .fcSendto` *`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 non-array or string array variable. The maximum size for a string non-array is 255 bytes; for a string array, 1472.

The transmit mode (*`SENDMODE%`*) must be one of the following values:

`.soSdNrm`	`0`	Normal
`.soSdDnRt`	`4`	Bypass pathway control function

The protocol family (*`FAMILY%`*) must be 2, the value indicating the ARPA Internet protocols.

`.soINet`	`2`	ARPA Internet protocols

When specifying the value greater than 32767, describe in hexadecimal notation.

Example: PORT% = &h8000' Specify Port 32768

Return value: *`SENDSIZE%`* Number of bytes transmitted

Run-time errors:

Error code	Meaning
105h	Power-off detected
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.
241h	There is no connection pathway to the host.

Function #24 `.fcSSckOpt`
Set socket options

Syntax: `CALL "SOCKET.FN3" .fcSSckOpt` *`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%`*)		
`.soKepAliv`	2	Keep-alive timer enable/disable	`.soDisable`	0	Disabled
			`.soEnable`	1	Enabled

Option Number (*`OPTNAME%`*)		Description	Values for Option (*`OPTION`*)	Initial values
`.soSndBuff`	8	Transmit buffer size (byte)	1 to 8192	8192
`.soRcvBuff`	9	Receive buffer size (byte)	1 to 8192	8192
`.soMaxRT`	26	Retry count	0 to 32	12
`.soTIMEWAIT`	29	Status retaining period after closing TCP socket (seconds)	0 to 60	60
`.soRTODef`	30	Initial round trip time (ms)*	100 to 3000	3000
`.soRTOMin`	31	Minimum round trip time (ms)*	100 to 1000	100
`.soRTOMax`	32	Maximum round trip time (ms)*	100 to 60000	60000

* To be set in units of 100.

Run-time errors:

Error code	Meaning
105h	Power-off detected.
106h	An internal error has occurred in the TCP/IP module during data transmission.
107h	The TCP/IP module has not been initiated.
108h	The memory for the TCP/IP module has became insufficient during data transmission.
201h	Cannot set option after connection established
209h	Socket identifier is invalid.
216h	A parameter is invalid.

Function #25 .fcShutdwn
Shut down socket

Syntax: `CALL "SOCKET.FN3" .fcShutdwn` *`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:

`.soSdRecv`	`0`	Receive
`.soSdSend`	`1`	Transmit
`.soSdBoth`	`2`	Both

Return value: (None)

Run-time errors:

Error code	Meaning
`105h`	Power-off detected
`106h`	An internal error has occurred in the TCP/IP module during data transmission.
`107h`	The TCP/IP module has not been initiated.
`108h`	The memory for the TCP/IP module has became insufficient during data transmission.
`209h`	Socket identifier is invalid.
`216h`	A parameter is invalid.
`22Ah`	This option is not recognized at the specification level.

Function #26 .fcSocket
Create socket

Syntax: `CALL "SOCKET.FN3" .fcSocket` *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.

`.soINet`	2	ARPA Internet protocols

The socket type (*TYPE%*) must be one of the following values:

`.soStream`	1	Stream socket
`.soDGRam`	2	Datagram socket
`.soSoRaw`	3	RAW socket

The protocol layer (*PROTOCOL%*) must be one of the following values:

`.soICMP`	1	ICMP
`.soTCP`	6	TCP
`.soUDP`	17	UDP

Return value: *SOCKFD%* Socket identifier

Run-time errors:

Error code	Meaning
`105h`	Power-off detected.
`106h`	An internal error has occurred in the TCP/IP module during data transmission.
`107h`	The TCP/IP module has not been initiated.
`108h`	The memory for the TCP/IP module has became insufficient during data transmission.
`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 `.fcClose`

Close socket

Syntax: `CALL "SOCKET.FN3" .fcClose` *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
106h	An internal error has occurred in the TCP/IP module during data transmission.
107h	The TCP/IP module has not been initiated.
108h	The memory for the TCP/IP module has became insufficient during data transmission.
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 `.fcTSetup`
Specify TCP/IP communications pathway

Syntax: `CALL "SOCKET.FN3" .fcTSetup` *`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 one of the following values:

-	0	(Reserved for system)
-	2	(Reserved for system)
`.soDvCOM4`	3	COM4 (Bluetooth communications device) For models equipped with a Bluetooth communications device.

The link layer (*`LAYERMODE%`*) must be one of the following values:

`.soLyPPP`	0	PPP client
-	2	(Reserved for system)

Return value: *`INTERFACE%`* Communications pathway

Run-time errors:

Error code	Meaning
`100h`	Cannot specify communications pathway

Function #41. `fcTCnnSys`
Connect TCP/IP communications pathway with system settings

Syntax: `CALL "SOCKET.FN3" .fcTCnnSys` *`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
216h	A parameter is invalid.

Function #42 `.fcTCnnUsr`

Connect TCP/IP communications pathway with user settings

Syntax: `CALL "SOCKET.FN3" .fcTCnnUsr` *`INTERFACE%`*`,` *`IPADDRESS$`*`,` *`SUBNETMASK$`*`,` *`GATEWAY$`*`,` *`PPPMODE%`*`,` *`USERNAME$`*`,` *`PASSWORD$`*

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
PPPMODE%	PPP authentication procedure
USERNAME$	User name for PPP authentication
PASSWORD$	Password for PPP authentication

The PPP authentication procedure (*PPPMODE%*) must be one of the following values:

`.soPPPAuthNo`	`0`	None authentication
`.soPPPPAP`	`1`	PAP
`.soPPPCHAP`	`2`	CHAP

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
`216h`	A parameter is invalid.

Function #43 `.fcTDiscnn`

Disconnect TCP/IP communications pathway

Syntax: `CALL "SOCKET.FN3" .fcTDiscnn` *`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
`216h`	A parameter is invalid.

Function #44 `.fcTSysGet`
Get TCP/IP system settings

Syntax: `CALL "SOCKET.FN3" .fcTSysGet` *PARA%, data*

where *data* is *DATA%* or *DATA$*

Description: This function gets the current setting for the specified TCP/IP system settings.

Parameters: *PARA%* Item number

Return value: *data* Current setting for TCP/IP system settings (*DATA%*/*DATA$*)

Correspondence tables:

Item number (*PARA%*)		Description	Values for Setting (*DATA%*)		
.soPPPAuth	4	PPP authentication procedure	.soPPPAuthNo	0	None authentication
			.soPPPPAP	1	PAP
			.soPPPCHAP	2	CHAP
.soDvGet	100	Communications device	-	0	(Reserved for system)
			-	2	(Reserved for system)
			.soDvCOM4	3	COM4
.soLyGet	200	Link layer	.soLyPPP	0	PPP
			-	2	(Reserved for system)

Item number (*PARA%*)		Description	Values for Setting (*DATA$*)
.soPmIPAdr	1	IP address	Character string in dotted quad notation, maximum 15 bytes
.soPmNtMsk	2	Subnet mask	Character string in dotted quad notation, maximum 15 bytes
.soPmDGWay	3	Default gateway	Character string in dotted quad notation, maximum 15 bytes
.soPPPUser	5	User name for PPP authentication	Character string, maximum 15 bytes
.soPPPPw	6	Password for PPP authentication	Character string, maximum 15 bytes

Function #45 `.fcTSysSet`
Set TCP/IP system settings

Syntax: `CALL "SOCKET.FN3" .fcTSysSet` *`PARA%, data`*

where *`data`* is *`DATA%`* or *`DATA$`*

Description: This function sets the specified TCP/IP system settings to the new value.

Parameters:

`PARA%`	Item number
data	New setting for TCP/IP system settings *(DATA%/DATA$*)

Return value: (None)

Correspondence tables:

Refer to Table under function #44.

Function #46 `.fcTStsGet`
Get TCP socket status

Syntax: `CALL "SOCKET.FN3" .fcTStsGet` *`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.

`.soStRmtCl`	`&h0020`	Close socket from the opposite end (FIN received)

Note: Specifying an invalid state sometimes stops processing.

`TIMEOUT%` must be one of the following values:

`.soNoWait`	`-1`	No timeout
`.soNotTOut`	`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
`209h`	Socket identifier is invalid.
`216h`	A parameter is invalid.

17.6 FTP Library (FTP.FN3)

17.6.1 Overview

■ String Variables

The following are the string variables used by this library together with their memory requirements.

Description	Variable name	Size in bytes
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 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
.fcFTPOpnS	1	Open FTP client session with system settings	USER/PASS
.fcFTPOpnU	2	Open FTP client session with user settings	USER/PASS
.fcFTPClos	3	Close FTP client session	---
.fcPWD	4	Get current directory on FTP server	PWD
.fcCWD	5	Change current directory on FTP server	CWD
.fcRETR	6	Download file from FTP server	RETR
.fcSTOR	7	Upload file to FTP server	STOR/APPE
.fcFSysGet	8	Get FTP system settings	---
.fcFSysSet	9	Change FTP system settings	---
.fcRNFR	10	Change file name on FTP server	RNFR/RNTO
.fcPORT	11	Set port number for file transfer	PORT
.fcDELE	12	Delete file from FTP server	DELE

Refer to 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. (Refer to 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.
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.
425	Can't open data connection.

Reply Codes	Description
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.

17.6.2 Detailed Function Specifications

Function #1 `.fcFTPOpnS`
Open FTP client session with system settings

Syntax: `CALL "FTP.FN3" .fcFTPOpnS` *`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
`106h`	An internal error has occurred in the TCP/IP module during data transmission.
`107h`	The TCP/IP module has not been initiated.
`108h`	The memory for the TCP/IP module has became insufficient during data transmission.
`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.
`23Ah`	The specified TCP socket has been closed.
`23Ch`	The connection attempt has timed out.
`293h`	The problem occurred on the communication pathway.

Function #2 `.fcFTPOpnU`

Open FTP client session with user settings

Syntax: `CALL "FTP.FN3" .fcFTPOpnU` *`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
`106h`	An internal error has occurred in the TCP/IP module during data transmission.
`107h`	The TCP/IP module has not been initiated.
`108h`	The memory for the TCP/IP module has became insufficient during data transmission.
`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.
`23Ah`	The specified TCP socket has been closed.
`23Ch`	The connection attempt has timed out.
`293h`	The problem occurred on the communication pathway.

Function #3 `.fcFTPClos`

Close FTP client session

Syntax: `CALL "FTP.FN3" .fcFTPClos` *`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
`106h`	An internal error has occurred in the TCP/IP module during data transmission.
`107h`	The TCP/IP module has not been initiated.
`108h`	The memory for the TCP/IP module has became insufficient during data transmission.
`110h`	Response other than 2XX received
`216h`	The FTP client handle is invalid.
`239h`	The specified socket is not connected.

Function #4 `.fcPWD`
Get current directory on FTP server

Syntax: `CALL "FTP.FN3" .fcPWD` *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
106h	An internal error has occurred in the TCP/IP module during data transmission.
107h	The TCP/IP module has not been initiated.
108h	The memory for the TCP/IP module has became insufficient during data transmission.
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 `.fcCWD`
Change current directory on FTP server

Syntax: `CALL "FTP.FN3" .fcCWD` *`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
`106h`	An internal error has occurred in the TCP/IP module during data transmission.
`107h`	The TCP/IP module has not been initiated.
`108h`	The memory for the TCP/IP module has became insufficient during data transmission.
`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 `.fcRETR`

Download file from FTP server

Syntax: `CALL "FTP.FN3" .fcRETR` *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 BHT, 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

`.ftCRLF`	0	CR-LF combination (Treat CR-LF combinations as delimiters. Use this value when the data file delimits records with CR-LF combinations.)
`.ftCR`	1	LF (Treat LFs as delimiters. Use this value when the data file delimits records with LFs.)
`.ftLF`	2	CR (Treat CRs as delimiters. Use this value when the data file delimits records with CRs.)
`.ftNONE`	3	None Use this value when the data file does not delimit records.

CRLF.MODE% Treatment of line delimiters in records and trailing spaces in fields

Note: *CRLF.MODE%* will be ignored for files except data files.

`.ftRcdSepa`	0	Treat line delimiters in records as SEPARATORS. TRIM trailing spaces in fields.

.ftRcdData	1	Treat line delimiters in records as DATA. TRIM trailing spaces in fields.
.ftLspDel	10	Treat line delimiters in records as SEPARATORS. RETAIN trailing spaces in fields.
.ftLspData	11	Treat line delimiters in 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

.ftNotDisp	0	Disable
.ftDisp	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 BHT
                                    'Same as on server
CRLF.TYPE% = .ftCR                  'Server line delimiter: LF
CRLF.MODE% = .ftRcdSepa             'Data composition
                                    'There are no line delimiters in the data.
FLD$ = "3, 2, 1"                    'Field lengths: 3, 2, 1
CALL "FTP.FN3" .fcRETR 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 BHT
                                    'Same as on server
CRLF.TYPE% = .ftCRLF                'Server line delimiter: CR-LF combination
```

```
CRLF.MODE% = .ftRcdSepa               'Data composition: Will be ignored for
                                      'files except data files
DISP.MODE% = .ftDisp                  'Enable progress display
CALL "FTP.FN3" .fcRETR 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
106h	An internal error has occurred in the TCP/IP module during data transmission.
107h	The TCP/IP module has not been initiated.
108h	The memory for the TCP/IP module has became insufficient during data transmission.
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 .fcSTOR

Upload file to FTP server

Syntax: CALL "FTP.FN3" .fcSTOR *FTPHANDLE%, SERV.FNAME$, CLNT.FNAME$, CRLF.TYPE%, UP.MODE%, REPLY% [,DISP.MODE%]*

Description: This function uploads, from the BHT 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 (Refer to description under function #6 above.)

UP.MODE% Flag controlling treatment of existing files

.ftUpSTOR	0	Overwrite existing file
.ftUpAPPE	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 Refer to the *DISP.MODE%* under function #6.

Return value: *REPLY%* Server response to FTP command

Example: Uploading data file

```
CLNT.FNAME$ = "MASTER1.DAT"          'Name of file on BHT
SERV.FNAME$ = ""                     'Name on server
                                     'Same as on BHT
CRLF.TYPE% = .ftCRLF                 'Server line delimiter: CR-LF combination
UP.MODE% = .ftUpAPPE                 'Upload mode: Append
CALL "FTP.FN3" .fcSTOR 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 BHT
SERV.FNAME$ = ""                    'Name on server
                                    'Same as on BHT
CRLF.TYPE% = .ftCRLF                'Server line delimiter: CR-LF combination
UP.MODE% = .ftUpSTOR                'Upload mode: Overwrite
DISP.MODE% = .ftDisp                'Enable progress display
CALL "FTP.FN3" .fcSTOR 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
106h	An internal error has occurred in the TCP/IP module during data transmission.
107h	The TCP/IP module has not been initiated.
108h	The memory for the TCP/IP module has became insufficient during data transmission.
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 `.fcFSysGet`
Get FTP system settings

Syntax: `CALL "FTP.FN3" .fcFSysGet` *`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%`* Item number

Return value: *`ftp.para`* Current setting for FTP system settings of type integer/string (*`FTP.PARA%`*/*`FTP.PARA$`*)

Correspondence tables:

Item number (*PARA%*)		Description	Values for Setting (*FTP.PARA%*)
`.ftCRLFTyp`	5	Line delimiter	0 (CR-LF), 1 (LF), 2 (CR), 3 (None)
`.ftCRLFMd`	6	Treatment of line delimiters inside records	0 (separators), 1 (data)
`.ftUpMd`	7	Upload mode	0 (overwrite), 1 (append)
`.ftDispMd`	8	Progress display	0 (disable), 1 (enable)

Item number (*PARA%*)		Description	Values for Setting (*FTP.PARA$*)
`.ftSrvIP`	1	IP address for FTP server	Character string in dotted quad notation, maximum 15 bytes
`.ftUsrNm`	2	User name for FTP authentication	Character string, maximum of 16 bytes
`.ftPswd`	3	Password for FTP authentication	Character string, maximum of 16 bytes
`.ftDefDir`	4	Initial directory on FTP server	Character string, maximum of 63 bytes

Function #9 `.fcFSysSet`
Change FTP system settings

Syntax: `CALL "FTP.FN3" .fcFSysSet` *`PARA%, ftp.para`*
where *`ftp.para`* is *`FTP.PARA%`* or *`FTP.PARA$`*

Description: This function changes the specified FTP system settings to the new value.

Parameters:

`PARA%`	Item number
`ftp.para`	New setting for FTP system settings of type integer/string (*`FTP.PARA%/FTP.PARA$`*)

Return value: (None)

Correspondence tables:

Refer to Table under .fcFSysGet.

Function #10 .fcRNFR
Change file name on FTP server

Syntax: `CALL "FTP.FN3" .fcRNFR` *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
106h	An internal error has occurred in the TCP/IP module during data transmission.
107h	The TCP/IP module has not been initiated.
108h	The memory for the TCP/IP module has became insufficient during data transmission.
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 .fcPORT

Set port number for file transfer

Syntax: `CALL "FTP.FN3" .fcPORT` *FTPHANDLE%, PORT%*

Description: This function sets a port number specified by *PORT%* for file transfer.

Parameters:

FTPHANDLE%	FTP client handle
PORT%	Port number

When specifying the value greater than 32767, describe in hexadecimal notation.

Example: PORT% = &h8000' Specify Port 32768

Return value: (None)

Run-time errors:

Error code	Meaning
105h	Power-off detected
106h	An internal error has occurred in the TCP/IP module during data transmission.
107h	The TCP/IP module has not been initiated.
108h	The memory for the TCP/IP module has became insufficient during data transmission.
216h	The FTP client handle is invalid.
239h	The specified socket is not connected.
295h	There is no user for login request.

Function #12 .fcDELE

Delete file from FTP server

Syntax: `CALL "FTP.FN3" .fcDELE` *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
106h	An internal error has occurred in the TCP/IP module during data transmission.
107h	The TCP/IP module has not been initiated.
108h	The memory for the TCP/IP module has became insufficient during data transmission.
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.

Chapter 18

Bluetooth (BHTs with Bluetooth communications device)

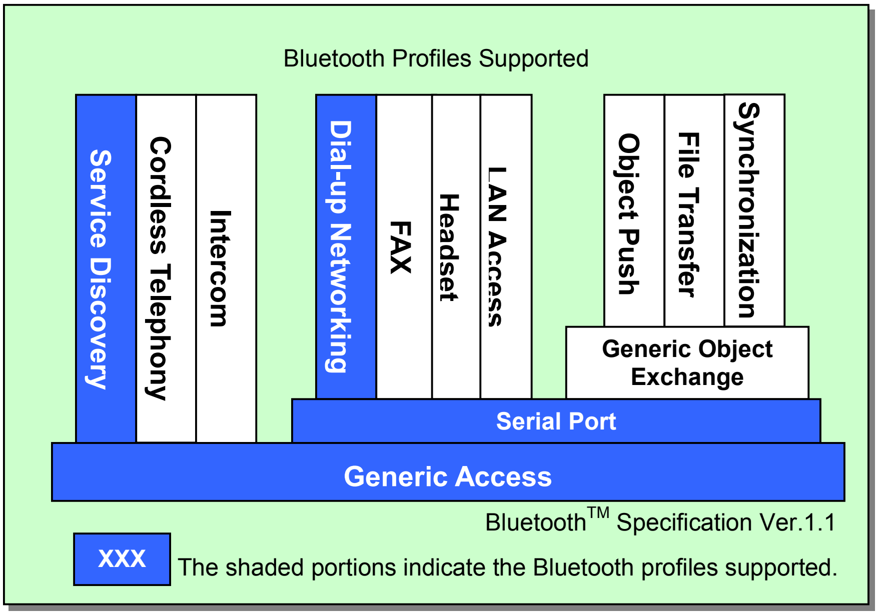
CONTENTS

18.1 Bluetooth Communications

18.1.1 Introduction

The BHT supports the following profiles based on the Bluetooth™ Specification Ver.1.1.

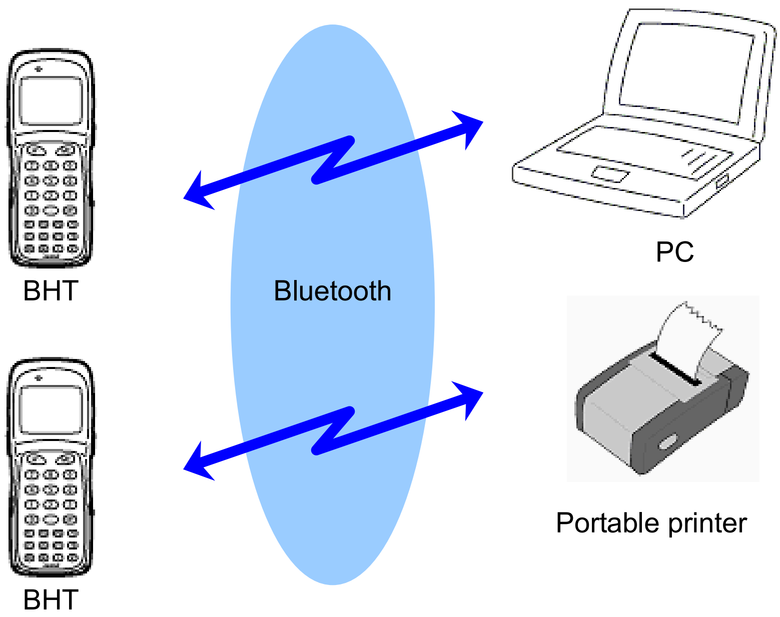
- The Generic Access Profile for discovering accessible Bluetooth devices in the vicinity
- The Serial Port Profile for RS232 (or similar) serial cable emulation through a virtual serial port
- The Dial-up Networking Profile for accessing the Internet via a modem or other device supporting dial-up access
- The Service Discovery Application Profile for querying and browsing for services offered by another Bluetooth device.



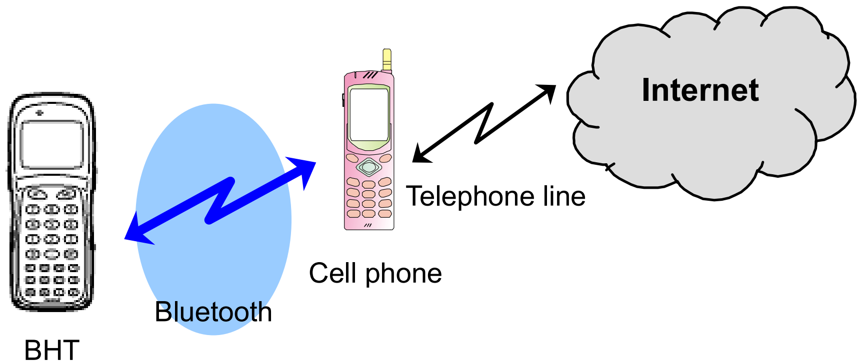
18.1.2 System Components

The following figures give examples of Bluetooth networks using the BHT. For further details, refer to the BHT User's Manual.

■ **Virtual Serial Link with PC or Portable Printer**

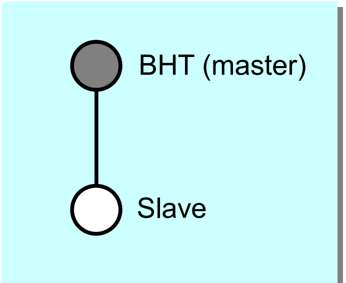


■ **Connecting to the Internet via a Cell Phone**

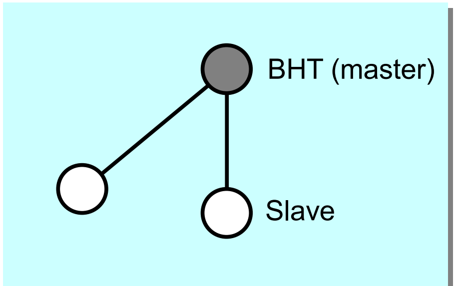


NOTE

• The BHT does not support multiple simultaneous links (Piconet.) As master, the BHT supports only one slave at a time.

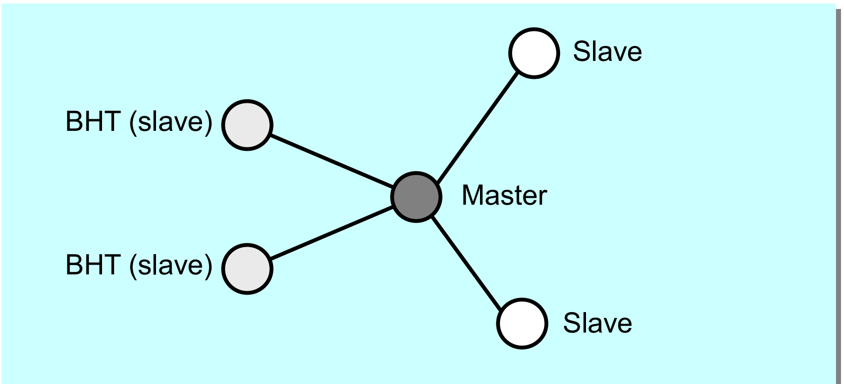


Available



Not Available

• As a slave, however, the BHT can connect to a master supporting multiple simultaneous links (Piconet.)



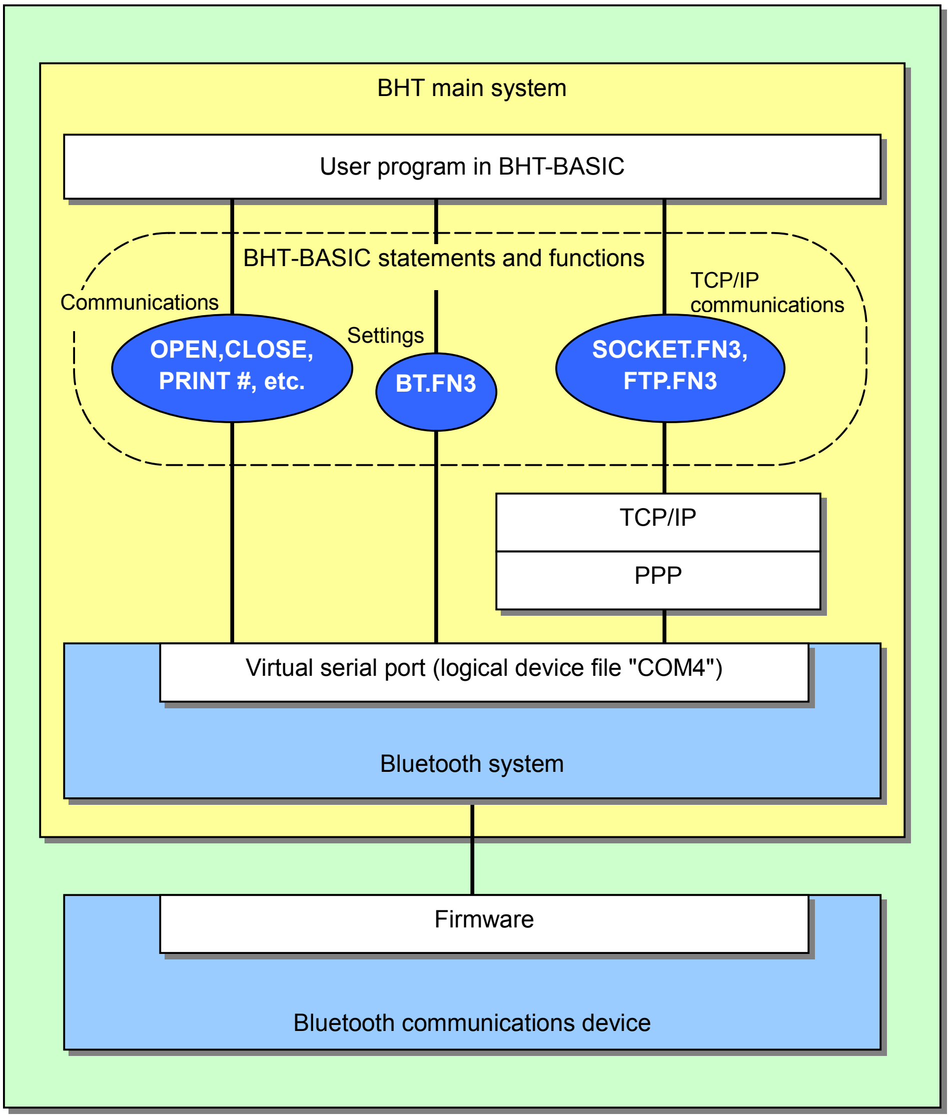
Available

18.2 Programming Overview

18.2.1 Software Components

The BHT system consists of the BHT main system and Bluetooth communications device. The former executes user programs and the latter performs Bluetooth communications.

User programs use the logical communications device file "COM4" to control the Bluetooth communications device.



18.2.2 Statements and Functions Used

Bluetooth communications uses the following statements and functions.

(1) Statements and functions

Refer to Section 18.3, "Bluetooth Statements and Functions."

(2) Bluetooth communications device control extended function (`BT.FN3`)

Refer to Section 18.4, "Bluetooth Extended Functions (`BT.FN3`)."

(3) Socket library for TCP/IP data transfer (`SOCKET.FN3`)

Refer to Section 17.5, "Socket Library (`SOCKET.FN3`)."

(4) FTP library for file transfer (`FTP.FN3`)

Refer to Section 17.6, "FTP Library (`FTP.FN3`)."

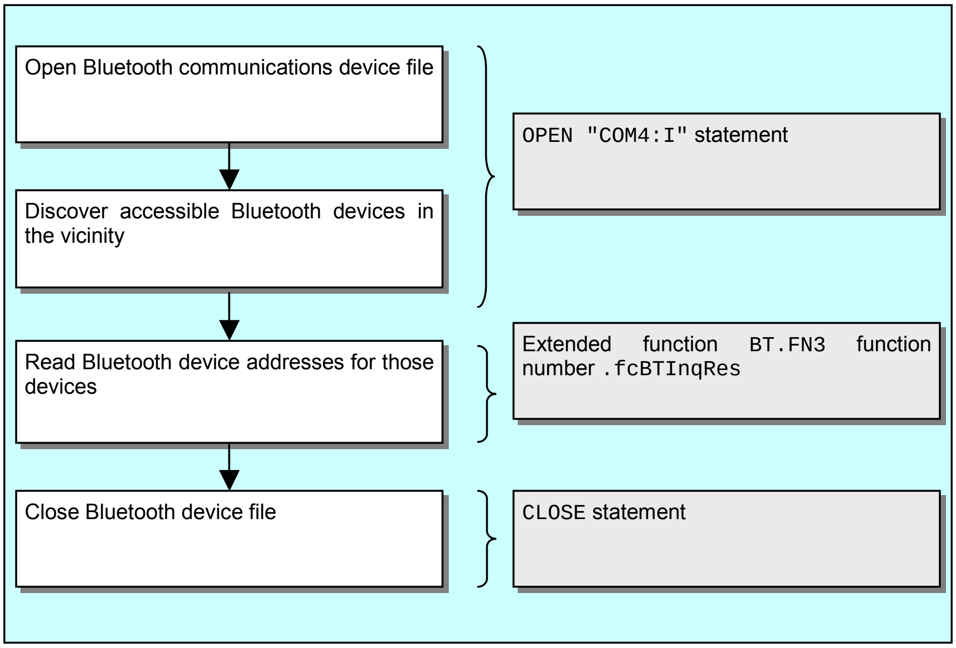
18.2.3 Programming Procedures

18.2.3.1 Discovering Accessible Remote Devices in the Vicinity (Inquiry)

The BHT supports the Generic Access Profile for discovering accessible Bluetooth devices in the vicinity.

Connecting to a remote device as master requires specifying the Bluetooth device address for that device. If that address is unknown, the BHT must first determine the addresses of accessible Bluetooth devices in the vicinity. If that address is already known, however, the user program can skip this step.

The following is the procedure for discovering remote devices.



Remote device discovery has the following parameters.

- Device discovery timeout, in seconds
- Number of devices to discover

Remote device discovery continues until the specified time elapses, the BHT finds the specified number of remote devices, or the user presses the clear key.

There are two ways to specify the above parameters.

- Use the system settings
- Specify them in the `OPEN` statement

The user modifies the system settings with the system menu; the user program, with extended function `BT.FN3` function numbers `.fcBTSetVal`. For further details on the system menu, refer to the BHT User's Manual.

Specifying a parameter in the `OPEN` statement does not affect the system settings.

Given below are examples discovering accessible remote devices in the vicinity.

(a) Using the system settings

```
OPEN "COM4:I" AS #4
```

(b) Specifying parameters in the `OPEN` statement

```
OPEN "COM4:I,20,3" AS #4               ' Device discovery timeout: 20 seconds
                                       ' Number of devices to discover: 3
```

For further details on the `OPEN "COM4:"` statement and `BT.FN3` extended function, refer to Sections 18.3 "Bluetooth Statements and Functions" and 18.4 "Bluetooth Extended Functions (`BT.FN3`),"respectively.

(Example)

Allow 30 seconds for discovering accessible remote devices in the vicinity. Stop at 5.

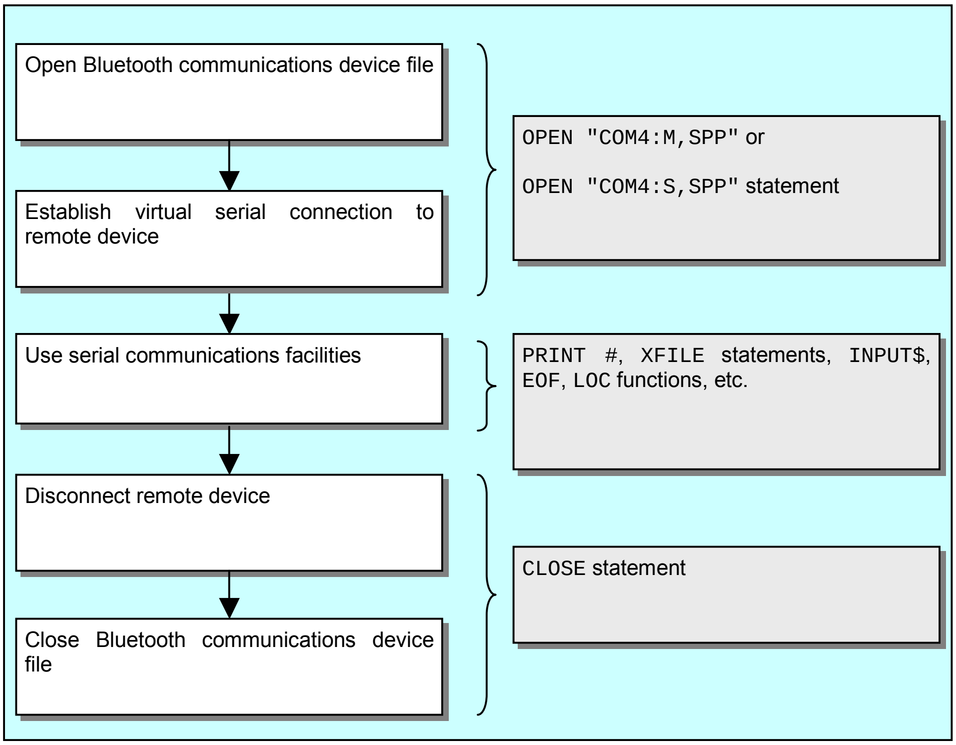
```
DEFREG bdaddr$(4)[12]
' Open Bluetooth communications device file
' Discover remote devices in the vicinity
OPEN "COM4:I,30,5" AS #4               ' Device discovery timeout: 30 seconds
                                       ' Number of devices to discover: 5
CALL "BT.FN3" .fcBTInqRes num%,bdaddr$()' Read discovery results
' Close Bluetooth communications device file
CLOSE #4
```

18.2.3.2 Serial Link with Remote Device

The BHT supports the Serial Port Profile for RS232 (or similar) serial cable emulation through a virtual serial port.

The BHT establishes a connection to an emulated serial port (or equivalent) in a remote device for serial communications. After connection, the interface is similar to the IrDA and direct-connect interfaces, using, for example, BHT-BASIC `PRINT #` statements for output and `INPUT$` function calls for input.

The following is the procedure for using such a serial link.



Remote device connections have the following parameters.

- Bluetooth device address for remote device (if BHT is master)
- Bluetooth passkey for master (or slave)
- Connection timeout, in seconds, for master (or slave)
- Security mode for master (or slave)

The BHT specifies master or slave operation when it opens the connection.

If it specifies master operation, the Bluetooth communications device automatically connects to the specified slave device. Otherwise, the Bluetooth communications device waits for a call from a master before connecting.

For further details on parameters, refer to the BHT User's Manual.

There are two ways to specify the above parameters.

- Use the system settings
- Specify them in the `OPEN` statement

The user modifies the system settings with the system menu; the user program, with extended function `BT.FN3` function numbers `.fcBTSetVal` and `.fcBTSetStr`. For further details on the system menu, refer to the BHT User's Manual.

Specifying a parameter in the `OPEN` statement does not affect the system settings.

Given below are examples connecting to the remote device as master.

(a) Using the system settings

```
OPEN "COM4:M,SPP" AS #4
```

(b) Specifying parameters in the `OPEN` statement

```
OPEN "COM4:M,SPP,112233AABBCC,BHT,30,2" AS #4
                                        ' Address for remote device:
                                        '   "11:22:33:AA:BB:CC"
                                        ' Bluetooth passkey: BHT
                                        ' Connection timeout: 30 seconds
                                        ' Security mode: service level
```

For further details on `OPEN "COM4:"` statements, refer to Section 18.3 "Bluetooth Statements and Functions."

(Example)

Connect as master via virtual serial port to the remote device and transfer data in both directions.

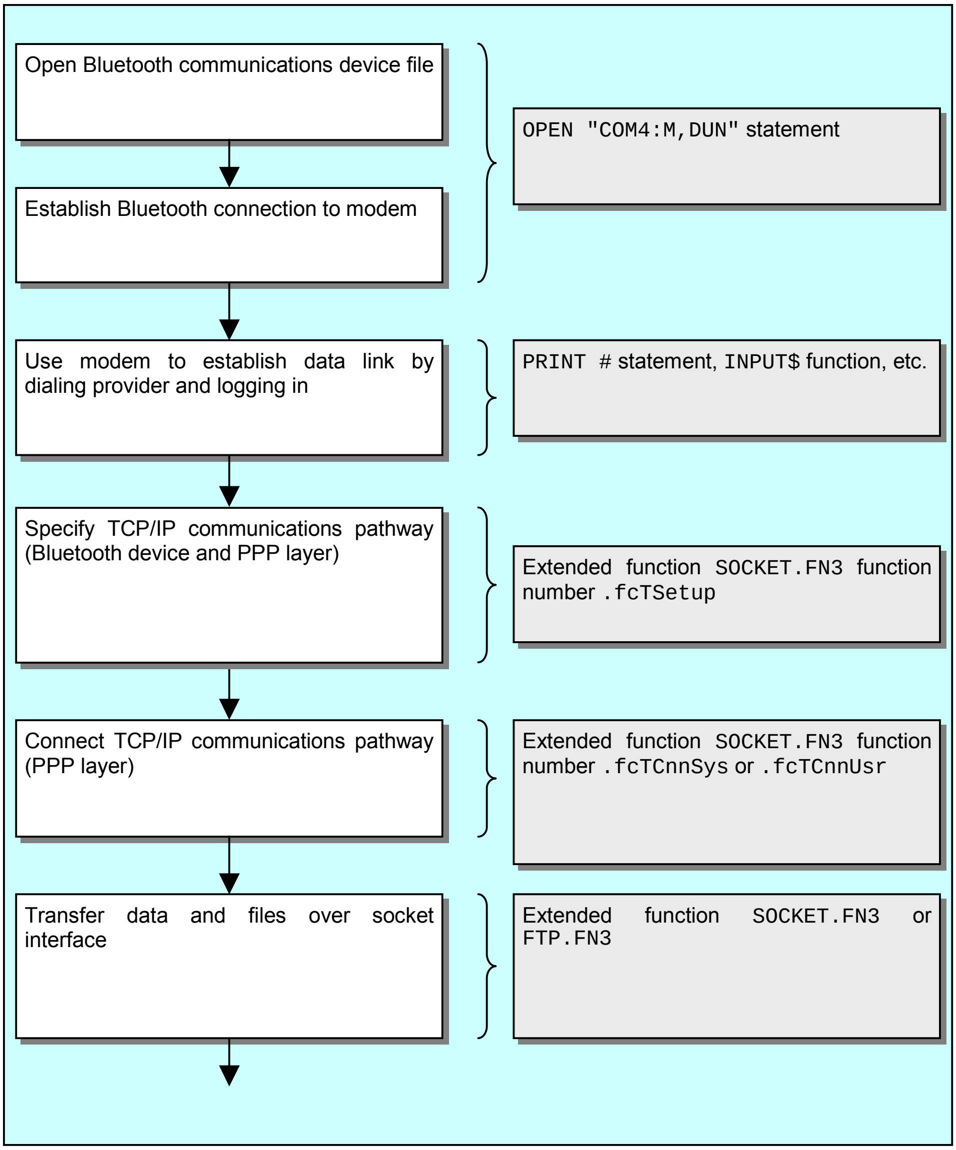
```
DIM sendbuff$[255]                      ' Allocate transmit buffer
DIM recvbuff$[255]                      ' Allocate receive buffer
' Create data to transmit
sendbuff$ = "ABCDEFG"                   ' Data to transmit = "ABCDEFG"
sendbuff$ = sendbuff$ + BCC$(sendbuff$,2)
                                        ' Add block check character
' Open Bluetooth communications device file
' Connect to remote device at address "11:22:33:AA:BB:CC"
```

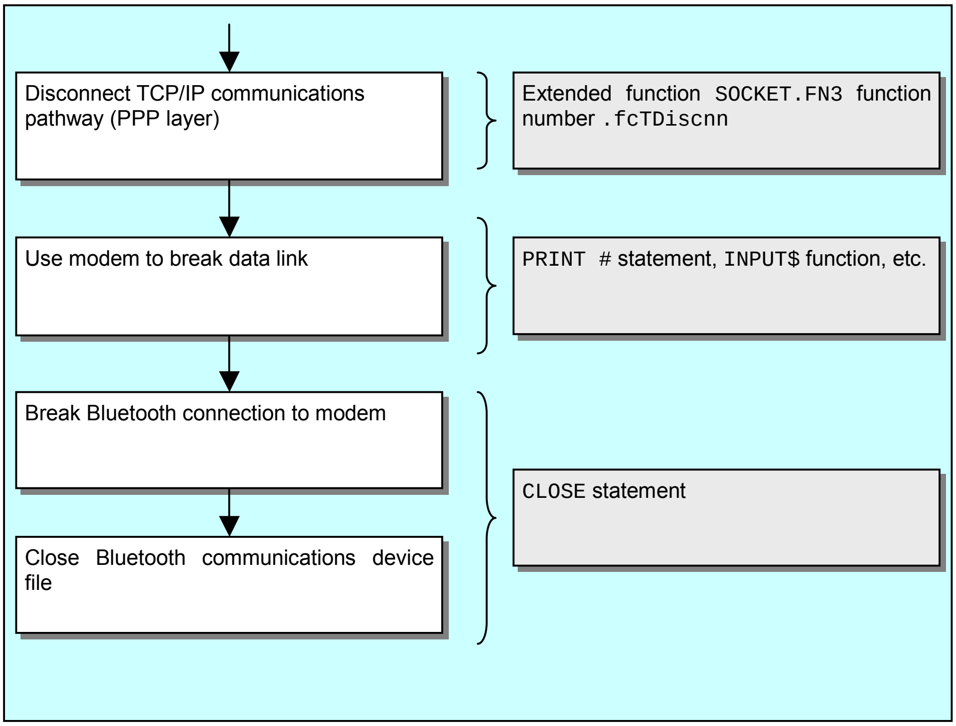
```
OPEN "COM4:M,SPP,112233AABBCC" AS #4    ' Address for remote device:
                                        '   11:22:33:AA:BB:CC"
' Use serial communications facilities
PRINT #4,sendbuff$;                     ' Transmit data

'   Read data received
'     For details about reading data received,
'     refer to Section 18.2.4.2 "Reading data received in serial communications."

' Disconnect remote device
' Close Bluetooth communications device file
CLOSE #4
```

18.2.3.3 Dial-Up Networking via Remote Device

The BHT supports the Dial-up Networking Profile for accessing the Internet via a modem or other device supporting dial-up access. The BHT uses a Bluetooth connection to control dial-up Internet access by the modem inside a cell phone, base station, or other suitably equipped device.

The following is the procedure for connecting to the Internet with such a modem and using TCP/IP communications.





Dial-up networking connections have the following parameters.

- Bluetooth device address for remote device
- Master Bluetooth passkey
- Master connection timeout in seconds
- Master security mode

For further details on parameters, refer to the BHT User's Manual.

There are two ways to specify the above parameters.

- Use the system settings
- Specify them in the `OPEN` statement

The user modifies the system settings with the system menu; the user program, with extended function `BT.FN3` function numbers `.fcBTSetVal` and `.fcBTSetStr`. For further details on the system menu, refer to the BHT User's Manual.

Specifying a parameter in the `OPEN` statement does not affect the system settings.

Given below are examples connecting to the Internet using a cell phone.

(a) Using the system settings

```
OPEN "COM4:M,DUN" AS #4
```

(b) Specifying parameters in the `OPEN` statement

```
OPEN "COM4:M,DUN,112233AABBCC,BHT,30,2" AS #4
                                        ' Address for remote device:
                                        '   11:22:33:AA:BB:CC"
                                        ' Bluetooth passkey: BHT
                                        ' Connection timeout: 30 seconds
                                        ' Security mode: service level
```

For further details on `OPEN "COM4:"` statement, refer to Section 18.3 "Bluetooth Statements and Functions."

(Example)

Connect to the Internet using a cell phone and transfer data and files over socket interface.

The cell phone has the following specifications.

Dial command	: "ATDT" + telephone number
Connect message	: "CONNECT"
Escape command	: "+++"
Disconnect command	: "ATH"
Reply message	: "OK"

```
' Open Bluetooth communications device file
' Connect to cell phone with Bluetooth device at address "11:22:33:AA:BB:CC"
    OPEN "COM4:M,DUN,112233AABBCC" AS #4  ' Address for remote device:
                                          '   "11:22:33:AA:BB:CC"
' Establish data link
    PRINT #4, "ATDT1234567890"            ' Dial provider (123-456-7890)
'   Wait for "CONNECT"
'     For details about the reading data received,
'     refer to Section 18.2.4.2 "Reading data received in serial communications."

' Specify TCP/IP communications pathway
    iftype% = .soDvCOM4                   ' Communications device: Bluetooth
    layermode% = .soLyPPP                 ' Link layer: PPP
    CALL "SOCKET.FN3" .fcTSetup iftype%,layermode%,Interface%
                                          ' Specify TCP/IP communications pathway
' Connect TCP/IP communications pathway
    ip$  = "192.168.0.125"                ' IP address for the BHT
    msk$ = "255.255.255.0"                ' Subnet mask
    gw$  = "0.0.0.0"                      ' Default gateway
```

```
    auh% = .soPPPPAP                    ' PPP authentication procedure: PAP
    usr$ = "USER"                       ' User name for PPP authentication
    psw$ = "PASSWORD"                   ' Password for PPP authentication
    CALL "SOCKET.FN3" .fcTCnnUsr Interface%,ip$,msk$,gw$,auh%,usr$,psw$

' Data and file transfers over socket interface
    'Omitted

' Disconnect TCP/IP communications pathway
    CALL "SOCKET.FN3" .fcTDiscnn Interface%

' Disconnect data link
    PRINT #4, "+++"                     ' Transmit escape command "+++"
    ' Wait for "OK"
    '   For details about the reading data received,
    '   refer to Section 18.2.4.2 "Reading data received in serial communications."
    PRINT #4, "ATH"                     ' Transmit disconnect command "ATH"
    ' Wait for "OK"
    '   For details about the reading data received,
    '   refer to Section 18.2.4.2 "Reading data received in serial communications."
' Disconnect modem and close Bluetooth communications device file
    CLOSE #4
    END
```

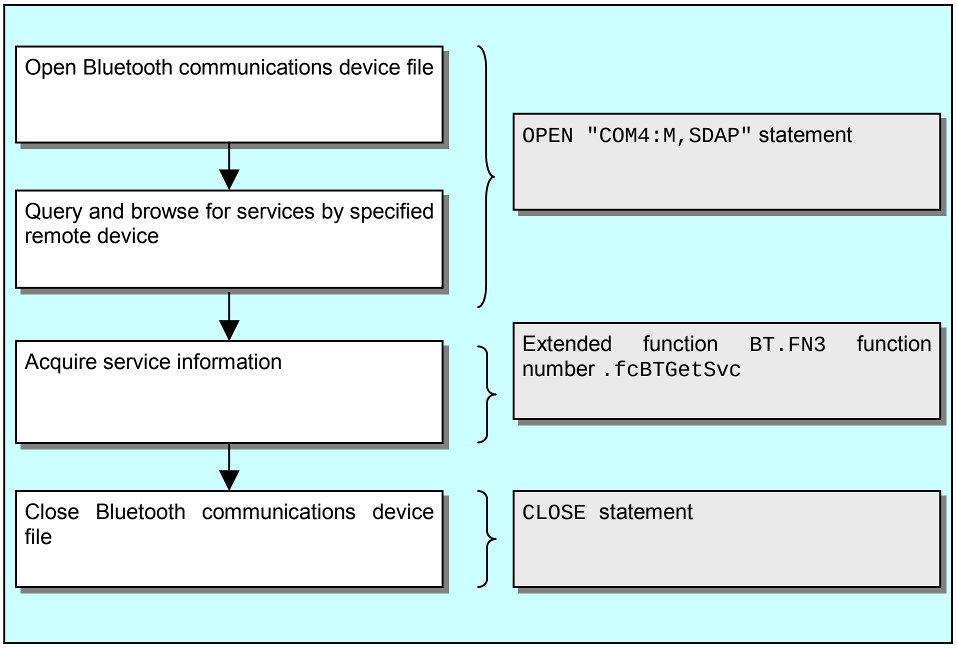
NOTE

- The above procedure assumes that the modem uses standard AT commands and response messages. Consult the modem's User's Manual for the strings used.
- This Dial-up Networking Profile does not support slave (GW) operation because the BHT does not have a built-in modem.

18.2.3.4 Service Discovery

The BHT supports the Service Discovery Application Profile for querying and browsing for services offered by another Bluetooth device.

The following is the procedure.



Service discovery has the following parameters.

- Bluetooth device address for remote device
- Master Bluetooth passkey
- Master connection (service discovery) timeout in seconds

For further details on parameters, refer to the BHT User's Manual.

There are two ways to specify the above parameters.

- Use the system settings
- Specify them in the `OPEN` statement

The user modifies the system settings with the system menu; the user program, with extended function `BT.FN3` function numbers `.fcBTSetVal` and `.fcBTSetStr`. For further details on the system menu, refer to the BHT User's Manual.

Specifying a parameter in the `OPEN` statement does not affect the system settings.

Given below are examples querying and browsing for services.

(a) Using the system settings

```
OPEN "COM4:M,SDAP" as #4
```

(b) Specifying parameters in the `OPEN` statement

```
OPEN "COM4:M,SDAP,112233AABBCC,BHT,60" as #4
                                        ' Address for remote device:
                                        '   11:22:33:AA:BB:CC"
                                        ' Bluetooth passkey: BHT
                                        ' Service discovery timeout: 60 seconds
```

For further details on the `OPEN "COM4:"` statement and `BT.FN3` extended function, refer to Sections 18.3 "Bluetooth Statements and Functions" and 18.4 "Bluetooth Extended Functions (`BT.FN3`)," respectively.

(Example)

Query and browse for services offered by Bluetooth device at address "11:22:33:AA:BB:CC."

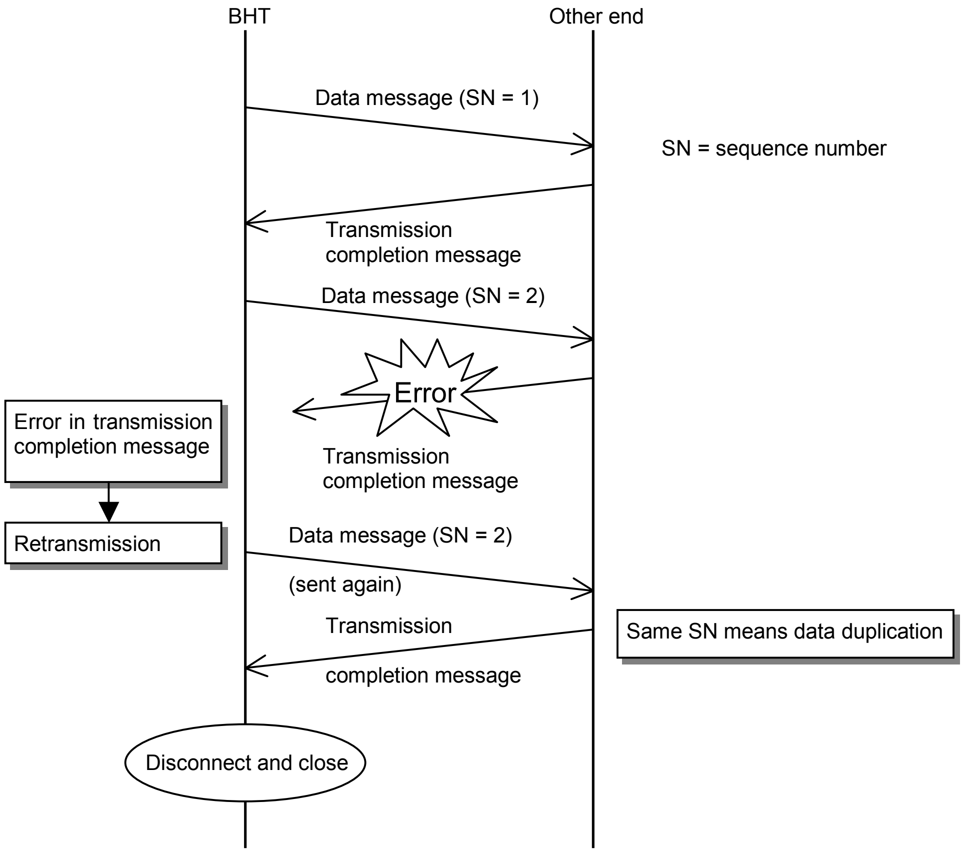
```
DIM sclass%(8)
DIM sname$(8)
' Open Bluetooth communications device file
' Query and browse for services offered by another Bluetooth device
'  at address"11:22:33:AA:BB:CC"
OPEN "COM4:M,SDAP,112233AABBCC" as #4   ' Address for remote device:
                                        '   "11:22:33:AA:BB:CC"
CALL "BT.FN3" .fcBTGetSvc num%, sclass%(), sname$()
                                        ' Acquire service discovery results
' Close Bluetooth communications device file
CLOSE #4
```

18.2.4 Programming Notes

18.2.4.1 Retransmission control in serial communications

Any system design using wireless communications must assume data losses due to line quality deterioration and data duplication due to delays during transmission. If the user program does not use the BHT protocol, the BHT-Ir protocol, or TCP/IP, it must implement its own protocol providing retransmission and flow control.

The following gives an example of such retransmission control for a user program.



The frequency of such communications errors varies considerably with the operating environment and usage conditions, so base the retransmission count and other parameters on thorough testing in a worst-case environment.

Extended function `BT.FN3` function number `.fcBTChkSnd` allows the user program to check whether all data messages transmitted actually reached the other end.

18.2.4.2 Reading data received in serial communications

We recommend that user programs always follow the approach shown below, setting a timeout and only reading data with `INPUT$` functions and the like when there is actual data in the receive buffer because there is every possibility of the direct approach hanging, waiting for data, due to disconnection of the remote device or motion out of communications range.

Note that extended function `BT.FN3` function number `.fcBTGetStt` is available for reading the connection status for the remote device.

(Example)

Connect via virtual serial port to the remote device and receive.

```
DIM recvbuff$[255]                    ' Allocate receive buffer
OPEN "COM4:M,SPP" AS #4               ' Open Bluetooth communications device file
recvbuff$ = ""                        ' Clear receive buffer
TIMEA = 50                            ' Receive wait timer: 5 seconds
WAIT 0,&h18                           ' Wait for data or timeout
IF LOC(#4) > 0 THEN                   ' If data received,
  WHILE LOC(#4) > 0                   ' read data received
    recvbuff$ = recvbuff$ + INPUT$(LOC(#4), #4)
    TIMEA = 5 : WAIT 0,&h10           ' Consider 500 ms with no input
                                      ' as indicating end of receive operation
  WEND
  PRINT "Receive ";recvbuff$          ' Display data received
ELSE                                  ' If no data received,
  PRINT "Receive timeout"             ' timeout
  CALL "BT.FN3" .fcBTGetStt STATUS%   ' Check current connection status
  IF STATUS%=2 OR STATUS%=3 THEN      ' If connected, receive again
    ' Retry receive
  ELSE                                ' If disconnected, close and connect again
    CLOSE #4
    ' Retry open and connect to remote device
  ENDIF
ENDIF
CLOSE #4
```

NOTE Do not use INPUT# or LINE INPUT# statement for reading data received. The INPUT# or LINE INPUT# statement waits for reception of CR (0Dh) or comma (,), so it cannot terminate in the case of data missing due to communications line error or disconnection of the communications line.

18.2.4.3 Resume Operation

Bluetooth communications does not support resume operation.

If the BHT shuts itself down due to low battery, etc, when the Bluetooth communications device file is opened, the results of Bluetooth-related statements and functions executed during shutdown are not assured so that coincidence between transmitted and received data is not assured. The solution is to use the BHT-protocol, BHT-Ir protocol, or TCP/IP or create protocols in user programs.

Extended function `BT.FN3` function number `.fcBTGetStt` allows the user program to check whether the BHT is turned off. If the current status is "Not connected. BHT power off," close the Bluetooth communications device file once and then open it.

For further details on extended function `BT.FN3`, refer to Section 18.4 "Bluetooth Extended Functions (`BT.FN3`)."

18.2.4.4 Power Supply Control

■ Power supply control of Bluetooth communications device

Closing the Bluetooth communications device file or switching to the power-saving mode while the Bluetooth communications device is not in use reduces the power consumption and extends the time that the BHT can be used between recharges.

Note, however, that the response is late because it takes several seconds to open the Bluetooth communications device file, connect to a remote device, and reach the state where communications is possible. The power-saving mode also introduces data communications delays. The developer must therefore tailor the use of these two approaches to match the intended application.

■ Power-saving mode

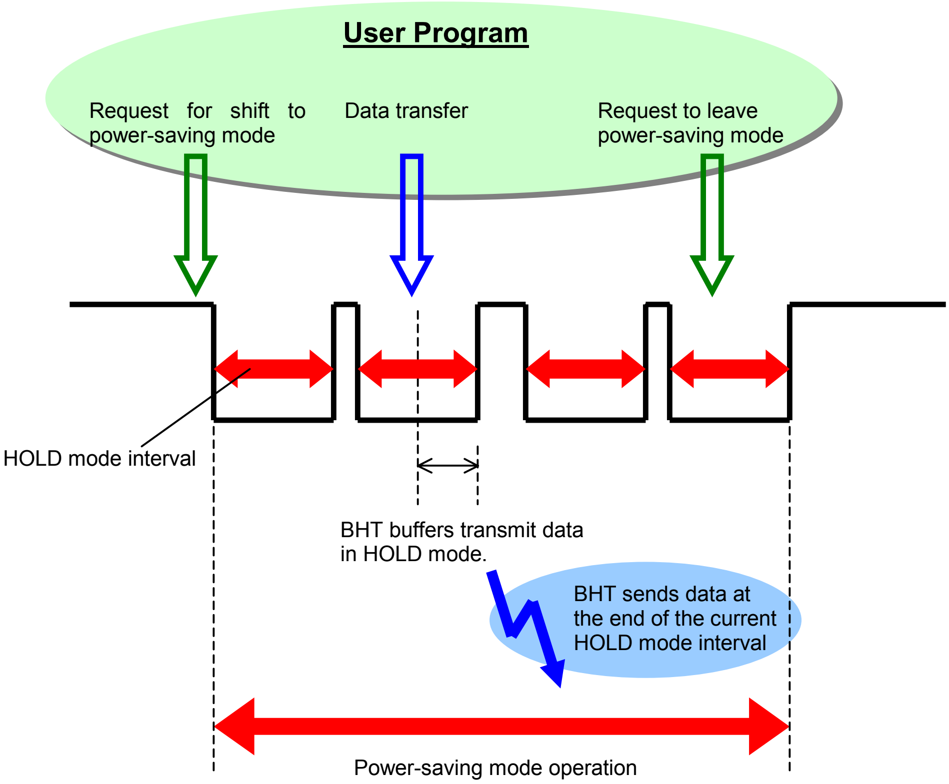
The BHT offers a power-saving mode with the following operation.

A request for shift to the power-saving mode in the user program shifts the BHT to the HOLD mode which suspends real-time transmission of any data and buffers the transmit data.

At the end of the HOLD interval specified by the user program, the BHT temporarily leaves the HOLD mode to check for data in the transmit buffer and for requests to leave the power-saving mode. If it finds neither, it immediately returns to the HOLD mode.

If there is data in the transmit buffer, the BHT sends all data in the transmit buffer and then returns to the HOLD mode. If the BHT finds a request to leave the power-saving mode, it goes out of the mode.

Power-saving mode control uses extended function `BT.FN3`. For further details, refer to Section 18.4 "Bluetooth Extended Functions (`BT.FN3`)."



NOTE

- Power-saving mode introduces delays.
  For operations involving real-time communications, we recommend that the power-saving mode be disabled. For operations using protocols for file transfer, etc., also disable the power-saving mode or set the HOLD mode interval that does not affect those transfer protocols. Otherwise, delays of data may cause protocol errors, resulting in communications errors.
- Note that the other end can also automatically enter HOLD mode during the HOLD mode interval, and that this end has no means to force it out of that mode.
- If the other end does not support the HOLD mode, there is no transition.

18.3 Bluetooth Statements and Functions

18.3.1 Overview

The following statements and functions are available for use with the Bluetooth communications device.

Statement or Function	Used to:
`OPEN "COM4:I"`	Open the Bluetooth communications device file in inquiry mode, discovering accessible remote devices in the vicinity
`OPEN "COM4:M"`	Open the Bluetooth communications device file with the BHT as master and connect to a slave
`OPEN "COM4:S"`	Open the Bluetooth communications device file with the BHT as a slave and wait for a master
`CLOSE`	Close the Bluetooth communications device file
`INPUT #`	Read data from the Bluetooth communications device file into specified variables
`LINE INPUT #`	Read data from the Bluetooth communications device file into a string variable
`PRINT #`	Write data to the Bluetooth communications device file
`WAIT`	Wait for a change in Bluetooth communications device file receive buffer status
`XFILE`	Transfer file using the specified communications protocol

Statement or Function	Used to:
`EOF`	Read whether there is data in the Bluetooth communications device file receive buffer
`LOC`	Read the number of bytes in the Bluetooth communications device file receive buffer
`LOF`	Read the number of bytes free in the Bluetooth communications device file receive buffer
`INPUT$`	Read data from the Bluetooth communications device file into a variable
`INP`	Read the status of Bluetooth communications device file receive buffer

18.3.2 Detailed Specifications

OPEN "COM4:I"

Open the Bluetooth communications device file in inquiry mode, discovering accessible remote devices in the vicinity

---

Syntax:

```
OPEN "COM4:I[, [discoverytime][,[ no.of.devices]]]" AS
[#]filenumber
```

Parameter:

discoverytime

Integer from 0 to 255.

no.of.devices

Integer from 0 to 8.

filenumber

A numeric expression which returns a value from 1 to 16.

Description:

This statement opens the Bluetooth communications device file in inquiry mode, discovering accessible remote devices in the vicinity.

Discovery continues until one of the following conditions is met.

- The BHT finds the specified number of devices.
- The specified time elapses.
- The user presses the clear key.

Note that the `OPEN` statement does not terminate until discovery is complete.

The extended function `BT.FN3` provides access to the discovery results--including any partial results obtained before the operation timed out or the user pressed the clear key.

■ **COM4**

This indicates the Bluetooth interface. Note that the BHT cannot open this communications device file concurrently with the IrDA interface or direct-connect interface.

■ **I**

This specifies opening in inquiry mode.

■ ***discoverytime***

This specifies the maximum interval to wait for responses from accessible remote devices. The unit is seconds; the range, 0 to 255. Note, however, that any value above 62 is rounded downward to produce a maximum discovery time of 62 seconds.

Specifying 0 opens the Bluetooth communications device file and skips device discovery.

Leaving this parameter blank specifies the use of the system setting.

Specifying a parameter does not affect the system setting.

■ ***no.of.devices***

This specifies an upper limit on the number of devices discovered. The OPEN statement terminates when it reaches this limit, regardless of the discovery time specified.

The range is 0 to 8. Specifying 0 sets the number to the maximum supported (8).

Leaving this parameter blank specifies the use of the system setting.

Specifying a parameter does not affect the system setting.

Syntax errors:

Refer to Chapter 14, "Statement Reference."

Run-time errors:

Error code	Meaning
105h	Power-off detected
600h	Failed to open a Bluetooth communications device file.

For error codes other than the above, refer to Chapter 14 "Statement Reference."

OPEN "COM4:M"

Open the Bluetooth communications device file with the BHT as master and connect to a slave

Syntax:

```
OPEN "COM4:M, serviceprofile [, [deviceaddress][,
[passkey][, [timeout][, [securitymode]]]]]" AS
[#]filenumber
```

Parameter:

serviceprofile

SDAP, SPP, or DUN.

deviceaddress

String of 12 hexadecimal digits.

passkey

Character string, Max. 16 bytes.

timeout

Integer from 1 to 255.

securitymode

Integer from 1 to 3.

filenumber

A numeric expression which returns a value from 1 to 16.

Description:

This statement opens the Bluetooth communications device file with the BHT as master and connects to a slave. (page)

All subsequent I/O and other operations involving the Bluetooth interface use the *filenumber*.

■ **COM4**

This indicates the Bluetooth interface. Note that the BHT cannot open this communications device file concurrently with the IrDA interface or direct-connect interface.

■ **M**

This specifies opening in master mode.

■ ***serviceprofile***

This specifies the service profile for the Bluetooth interface connection.

SDAP

Service Discovery Application Profile

The extended function BT.FN3 then provides access to the discovery results.

SPP

Serial Port Profile

DUN

Dial-up Networking Profile

■ ***deviceaddress***

This specifies the Bluetooth device address for the remote device as a string of 12 hexadecimal digits.

Leaving this parameter blank specifies the use of the system setting.

Specifying a parameter does not affect the system setting.

■ ***passkey***

This specifies the Bluetooth passkey (Bluetooth PIN), character string, Max. 16 bytes, for authentication between Bluetooth devices.

Leaving this parameter blank specifies the use of the system setting.

Specifying a parameter does not affect the system setting.

■ ***timeout***

This specifies a time limit for completing the operation. The unit is seconds; the range, 1 to 255.

Leaving this parameter blank specifies the use of the system setting.

Specifying a parameter does not affect the system setting.

■ ***securitymode***

This specifies the security mode for the connection, one of the following values.

Setting	Security Mode
1	Security mode 1 (nonsecure)
2	Security mode 2 (service level enforced security)
3	Security mode 3 (link level enforced security)

Leaving this parameter blank specifies the use of the system setting.

Specifying a parameter does not affect the system setting.

For further details on Bluetooth device address, Bluetooth passkey, and security mode, refer to the BHT User's Manual.

NOTE Pressing the clear key aborts the operation with run-time error 47h.
The operation aborts with run-time error 601h (630h for SDAP) if the specified device does not exist, if it is not able to accept the connection, or the Bluetooth passkey is incorrect. Check the remote device's status and the parameters and try again.
Service profile SDAP ignores the security mode setting and always uses 1, the " nonsecure" setting.

Syntax errors:

Refer to Chapter 14, "Statement Reference."

Run-time errors:

Error code	Meaning
`47h`	Abnormal end of communications or termination of communications by the Clear key
`105h`	Power-off detected
`600h`	Failed to open a Bluetooth communications device file.
`601h`	Failed to connect.
`602h`	Connection timed out.
`630h`	No services found.
`631h`	Service discovery timed out.

For error codes other than the above, refer to Chapter 14, "Statement Reference."

OPEN "COM4:S"

Open the Bluetooth communications device file with the BHT as a slave and wait for a master

Syntax:

```
OPEN "COM4:S, serviceprofile [, [passkey][, [timeout][,
[securitymode]]]]" AS [#]filenumber
```

Parameter:

serviceprofile

SPP

passkey

Character string, Max. 16 bytes.

timeout

Integer from 1 to 255.

securitymode

Integer from 1 to 3.

filenumber

A numeric expression which returns a value from 1 to 16.

Description:

This statement opens the Bluetooth communications device file with the BHT as a slave and wait for inquiries and connection requests from masters. (Inquiry Scan Enable and Page Scan Enable)

All subsequent I/O and other operations involving the Bluetooth interface use the *filenumber*.

■ **COM4**

This indicates the Bluetooth interface. Note that the BHT cannot open this communications device file concurrently with the IrDA interface or direct-connect interface.

■ **S**

This specifies opening in slave mode.

■ ***serviceprofile***

This specifies the service profile for the Bluetooth interface connection.

SPP

Serial Port Profile

■ ***passkey***

This specifies the Bluetooth passkey (Bluetooth PIN), character string, Max. 16 bytes, for authentication between Bluetooth devices.

Leaving this parameter blank specifies the use of the system setting.

Specifying a parameter does not affect the system setting.

■ ***timeout***

This specifies a time limit for completing the operation. The unit is seconds; the range, 1 to 255.

Leaving this parameter blank specifies the use of the system setting.

Specifying a parameter does not affect the system setting.

■ ***securitymode***

This specifies the security mode for the connection, one of the following values.

Setting	Security Mode
1	Security mode 1 (nonsecure)
2	Security mode 2 (service level enforced security)
3	Security mode 3 (link level enforced security)

Leaving this parameter blank specifies the use of the system setting.

Specifying a parameter does not affect the system setting.

For further details on Bluetooth device address, Bluetooth passkey, and security mode, refer to the BHT User's Manual.

NOTE Pressing the clear key aborts the operation with run-time error 47h.

Syntax errors:

Refer to Chapter 14, "Statement Reference."

Run-time errors:

Error code	Meaning
47h	Abnormal end of communications or termination of communications by the Clear key
105h	Power-off detected
600h	Failed to open a Bluetooth communications device file.
602h	Connection timed out.

For error codes other than the above, refer to Chapter 14, "Statement Reference."

`CLOSE` Close the Bluetooth communications device file

Refer to Chapter 14, "Statement Reference."

`INPUT #` Read data from the Bluetooth communications device file into specified variables

Refer to Chapter 14, "Statement Reference."

`LINE INPUT #` Read data from the Bluetooth communications device file into a string variable

Refer to Chapter 14, "Statement Reference."

`PRINT #` Write data to the Bluetooth communications device file

Syntax:

Refer to Chapter 14, "Statement Reference."

Parameter:

Refer to Chapter 14, "Statement Reference."

Description:

Refer to Chapter 14, "Statement Reference."

NOTE A `PRINT #` statement ends with the write of the data to the Bluetooth communications device file. It provides no guarantee that the data actually reached the other end. The user program must use either extended function `BT.FN3` function number `.fcBTChkSnd` or receive a confirmation message from the other end.

Syntax errors:

Refer to Chapter 14, "Statement Reference."

Run-time errors:

Error code	Meaning
`610h`	Bluetooth data link already disconnected.
`622h`	No response from Bluetooth interface.

For error codes other than the above, refer to Chapter 14, "Statement Reference."

`WAIT` Wait for a change in Bluetooth communications device file receive buffer status

Refer to Chapter 14, "Statement Reference."

`XFILE` Transfer file using the specified communications protocol

Refer to Chapter 14, "Statement Reference."

`EOF` Read whether there is data in the Bluetooth communications device file receive buffer

Refer to Chapter 14, "Statement Reference."

`LOC` Read the number of bytes in the Bluetooth communications device file receive buffer

Refer to Chapter 14, "Statement Reference."

`LOF` Read the number of bytes free in the Bluetooth communications device file receive buffer

Refer to Chapter 14, "Statement Reference."

`INPUT$` Read data from the Bluetooth communications device file into a variable

Refer to Chapter 14, "Statement Reference."

`INP` Read the status of Bluetooth communications device file receive buffer

Refer to Chapter 14, "Statement Reference."

18.4 Bluetooth Extended Functions (BT.FN3)

18.4.1 Overview

The Bluetooth extended functions (BT.FN3) used in a BHT-BASIC CALL statement reads or writes Bluetooth parameters and controls operation.

If Bluetooth communications device becomes no longer possible, a run-time error 105h may occur. In such a case, close the device file and then open again.

■ Function Number List of BT.FN3

Function number		Used to:
.fcBTGetVal	1	Read Bluetooth integer setting
.fcBTSetVal	2	Write Bluetooth integer setting
.fcBTGetStr	3	Read Bluetooth string setting
.fcBTSetStr	4	Write Bluetooth string setting
.fcBTSysVer	7	Read Bluetooth system version
.fcBTDevInf	8	Read Bluetooth device information
.fcBTRmtNam	9	Get remote device name
.fcBTInqRes	10	Read remote device discovery results
.fcBTRmtInf	11	Get Bluetooth device address for remote device
.fcBTGetStt	12	Read connection status
.fcBTGetLnk	13	Read authenticated Bluetooth device addresses
.fcBTClrLnk	14	Erase authenticated Bluetooth device addresses
.fcBTHold	15	Control power-saving mode
.fcBTChkSnd	20	Check data transmit result
.fcBTGetSvc	21	Read service discovery results

18.4.2 Detailed Specifications

Function #1 `.fcBTGetVal`
Read Bluetooth integer setting

Syntax: `CALL "BT.FN3" .fcBTGetVal PARA%,DATA%`

Description: This function reads the specified Bluetooth setting into the specified integer variable.

Parameters: *PARA%* Item number

Returned value: *DATA%* Integer read from the specified Bluetooth setting

Correspondence table:

Item number (*PARA%*)		parameter	Attribute[*1]	Parameter value (*DATA%*)	Initial value
`.btTTOInq`	1	Device discovery timeout	R/W	0 to 255 (unit: seconds)	10
`.btNumInq`	2	Number of devices to discover	R/W	0 to 8	0
`.btTOMst`	3	Master connection timeout	R/W	1 to 255 (unit: seconds)	30
`.btTOSlv`	4	Slave connection timeout	R/W	1 to 255 (unit: seconds)	255
`.btSecMst`	5	Master security mode	R/W	1 Nonsecure 2 Service level 3 Link level	1
`.btSecSlv`	6	Slave security mode	R/W	1 Nonsecure 2 Service level 3 Link level	1

[*1] R/W: Read and write possible

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range
`F0h`	Mismatch parameter number
`F1h`	Mismatch parameter type

Function #2 `.fcBTSetVal` Write Bluetooth integer setting

Syntax: `CALL "BT.FN3" .fcBTSetVal PARA%,DATA%`

Description: This function writes the specified value to the specified Bluetooth integer setting.

Parameters: *PARA%* Item number

DATA% New setting

Returned value: *(None)*

Correspondence table:

Refer to the correspondence table given in Function `.fcBTGetVal`.

Note: The new setting takes effect the next time that the Bluetooth communications device file is opened.

Run-time errors:

Error code	Meaning
05h	Parameter out of the range
F0h	Mismatch parameter number
F1h	Mismatch parameter type

Function #3 `.fcBTGetStr` Read Bluetooth string setting

Syntax: `CALL "BT.FN3" .fcBTGetStr PARA%,DATA$`

Description: This function reads the specified Bluetooth string setting into the specified string variable.

Parameters: *PARA%* Item number

Returned value: *DATA$* String read from the specified Bluetooth setting

Correspondence table:

Item number (*PARA%*)		parameter	Attribute[*1]	Parameter value (*DATA$*)	Initial value
`.btLocNam`	1	Bluetooth device name	WO	Character string, Max. 16 bytes	DENSO-BHT
`.btRmtAdr`	2	Bluetooth device address for remote device	R/W	String of 12 hexadecimal digits	000000000000
`.btKeyMst`	3	Master Bluetooth passkey	R/W	Character string, Max. 16 bytes	0000000000000000
`.btKeySlv`	4	Slave Bluetooth passkey	R/W	Character string, Max. 16 bytes	0000000000000000

[*1] WO: Write only
R/W: Read and write possible

Note: Function number `.fcBTDevInf` is available for reading the Bluetooth device name.

The Bluetooth passkey distinguishes between upper and lower case.

Run-time errors:

Error code	Meaning
`05h`	Parameter out of the range
`F0h`	Mismatch parameter number
`F1h`	Mismatch parameter type
`F2h`	Out of string variable space

Function #4 `.fcBTSetStr`
Write Bluetooth string setting

Syntax: `CALL "BT.FN3" .fcBTSetStr PARA%,DATA$`

Description: This function writes the specified value to the specified Bluetooth string setting.

Parameters: *PARA%* Item number

DATA$ New setting

Returned value: *(None)*

Correspondence table:

Refer to the correspondence table given in Function `.fcBTGetStr`.

Note: The new setting takes effect the next time that the Bluetooth communications device file is opened.

Run-time errors:

Error code	Meaning
05h	Parameter out of the range
F0h	Mismatch parameter number
F1h	Mismatch parameter type

Function #7 **.fcBTSysVer** Read Bluetooth system version

Syntax: `CALL "BT.FN3" .fcBTSysVer BTSYSVER$`

Description: This function reads the Bluetooth system version.

Parameters: *(None)*

Returned value: *BTSYSVER$* Bluetooth system version (fixed at 4 characters)

The user program must allocate at least 4 bytes to *BTSYSVER$*.

Run-time errors:

Error code	Meaning
05h	Parameter out of the range
F0h	Mismatch parameter number
F1h	Mismatch parameter type
F2h	Out of string variable space

Function #8 `.fcBTDevInf`
Read Bluetooth device information

Syntax: `CALL "BT.FN3" .fcBTDevInf PARA%,DATA$`

Description: This function reads Bluetooth device information.

Parameters: *PARA%* Item number

Returned value: *DATA$* Current Bluetooth information setting

Correspondence table:

Item number (*PARA%*)		parameter	Attribute[*1]	Parameter value (*DATA$*)
`.btFWVer`	1	Bluetooth device firmware version	RO	Character string, Max. 9 bytes
`.btDevAdr`	2	Bluetooth device address	RO	String of 12 hexadecimal digits
`.btDevNam`	3	Bluetooth device name	RO	Character string, Max. 16 bytes

[*1] RO: Read only

Note: The function should be executed after execution of `OPEN "COM4:"` statement.

Function number `.fcBTSetStr` is available for setting the Bluetooth device name.

Run-time errors:

Error code	Meaning
05h	Parameter out of the range
34h	Bad file name or number. (The Bluetooth communications device file is not opened)
F0h	Mismatch parameter number
F1h	Mismatch parameter type
F2h	Out of string variable space
105h	Power-off detected.
622h	No response from Bluetooth interface

Example:

Read and display Bluetooth device address.

```
OPEN "COM4:I,0" AS #4        ' Open Bluetooth communications device file
PARA% = .btDevAdr
CALL "BT.FN3" .fcBTDevInf PARA%,DATA$
                             ' Read Bluetooth device address
PRINT DATA$                  ' Display Bluetooth device address
CLOSE #4                     ' Close Bluetooth communications device file
```

Function #9 `.fcBTRmtNam`
Get remote device name

Syntax: `CALL "BT.FN3" .fcBTRmtNam BDADDR$,BDNAME$`

Description: This function gets the Bluetooth device name for the remote device at the specified Bluetooth address.

Parameters: *BDADDR$* Bluetooth device address (string of 12 hexadecimal digits)

Returned value: *BDNAME$* Device name (character string, Max. 248 bytes)

If the name is longer than the string length of *BDADDR$*, the interface discards the excess bytes.

Note: The function should be executed after execution of `OPEN "COM4:"` statement.

The operation aborts with run-time error 621h if the specified device does not exist or it is not able to accept the connection. Check the remote device's status and try again.

Run-time errors:

Error code	Meaning
05h	Parameter out of the range
34h	Bad file name or number. (The Bluetooth communications device file is not opened)
F0h	Mismatch parameter number
F1h	Mismatch parameter type
F2h	Out of string variable space
105h	Power-off detected.
621h	Failed to get remote device name.
622h	No response from Bluetooth interface

Example:

Get and display remote device name.

```
OPEN "COM4:I,0" AS #4        ' Open Bluetooth communications device file
BDADDR$ = "112233AABBCC"     ' Address for remote device:
                             '   "11:22:33:AA:BB:CC"
CALL "BT.FN3" .fcBTRmtNam BDADDR$,BDNAME$
                             ' Get remote device name
PRINT BDADDR$,BDNAME$        ' Display Bluetooth device address and name
CLOSE #4                     ' Close Bluetooth communications device file
```

Function #10 `.fcBTInqRes` Read remote device discovery results

Syntax: `CALL "BT.FN3" .fcBTInqRes NUM%,BDADDR$[()]`

Description: This function reads results of remote device discovery with a `OPEN "COM4:I"` statement.

Parameters: *(NONE)*

Returned value: *NUM%* Number of remote devices discovered (0 to 8)

BDADDR$[()]

Bluetooth device addresses (strings of 12 hexadecimal digits each) for remote device discovered.

NUM% gives the number of valid addresses in the array *BDADDR$*.

The user program must allocate at least 12 bytes to *BDADDR$*.

If *NUM%* is greater than 1, treat *BDADDR$* as an array variable.

If the number of devices discovered exceeds the number of *BDADDR$* entries, the interface stops when the array is full.

Note: The function should be executed after execution of `OPEN "COM4:I"` statement.

Run-time errors:

Error code	Meaning
05h	Parameter out of the range
34h	Bad file name or number. (The Bluetooth communications device file is not opened)
F0h	Mismatch parameter number
F1h	Mismatch parameter type
F2h	Out of string variable space
105h	Power-off detected.
622h	No response from Bluetooth interface

Example:

Discover remote devices and display results

```
DIM BDADDR$(7)[12]           ' Allocate space for 8 devices
OPEN "COM4:I,30,0" AS #4     ' Discover remote devices
CALL "BT.FN3" .fcBTInqRes NUM%,BDADDR$()
                             ' Read discovery results
CLOSE #4                     ' Close Bluetooth communications device file
FOR I%=0 TO NUM%-1
   PRINT BDADDR$(I%)         ' Display device address
NEXT
```

Function #11 `.fcBTRmtInf`
Get Bluetooth device address for remote device

Syntax: `CALL "BT.FN3" .fcBTRmtInf BDADDR$`

Description: This function gets the Bluetooth device address for the connected remote device.

Parameters: *(None)*

Returned value: *BDADDR$* Bluetooth device address (string of 12 hexadecimal digits) for connected remote device

The user program must allocate at least 12 bytes to *BDADDR$*.

Note: The function should be executed after execution of `OPEN "COM4:M"` or `OPEN "COM4:S"` statement.

Run-time errors:

Error code	Meaning
05h	Parameter out of the range
34h	Bad file name or number. (The Bluetooth communications device file is not opened)
F0h	Mismatch parameter number
F1h	Mismatch parameter type
F2h	Out of string variable space
105h	Power-off detected.
620h	Not connected to a remote device

Function #12 `.fcBTGetStt` Read connection status

Syntax: `CALL "BT.FN3" .fcBTGetStt STATUS%`

Description: This function reads the current connection status.

Parameters: *(None)*

Returned value: *STATUS%* Current connection status

STATUS%	Current Connection Status
0	Not connected.
1	Not connected. Connection broken by the other end.
2	Connected.
3	Connected (in power-saving mode.)
4	Not connected. BHT power off.

Note: If the BHT in connection with a remote device is disconnected by the device, "1" (Not connected. Connection broken by the other end) is returned to STATUS%. If the BHT is turned off, "4" (Not connected. BHT power off) is returned. In either of these cases, close the Bluetooth communications device file once and then open it.

Run-time errors:

Error code	Meaning
`F0h`	Mismatch parameter number
`F1h`	Mismatch parameter type

Function #13 `.fcBTGetLnk`

Read authenticated Bluetooth device addresses

Syntax: `CALL "BT.FN3" .fcBTGetLnk NUM%,BDADDR$[()]`

Description: This function reads the Bluetooth device addresses of authenticated remote devices.

Parameters: *(NONE)*

Returned value:

NUM% Number of authenticated remote devices (0 to 3)

BDADDR$[()]

Bluetooth device addresses (strings of 12 hexadecimal digits each) for authenticated remote devices.

NUM% gives the number of valid addresses in the array *BDADDR$*.

The user program must allocate at least 12 bytes to *BDADDR$*.

If *NUM%* is greater than 1, treat *BDADDR$* as an array variable.

If the number of devices detected exceeds the number of *BDADDR$* entries, the interface stops when the array is full.

Run-time errors:

Error code	Meaning
F0h	Mismatch parameter number
F1h	Mismatch parameter type
F2h	Out of string variable space

Function #14 `.fcBTClrLnk` Erase authenticated Bluetooth device addresses

Syntax: `CALL "BT.FN3" .fcBTClrLnk [BDADDR$]`

Description: This function erases the Bluetooth device addresses of authenticated remote devices.

Parameters: *BDADDR$[()]*

Authenticated Bluetooth device addresses (strings of 12 hexadecimal digits each) to erase.

Omitting the *BDADDR$* parameter erases the entire list.

Returned value: *(NONE)*

Note: Erasing a authenticated Bluetooth device address may make it impossible to connect to the corresponding remote device using security mode 3 (link level enforced security.) If this happens, try reconnecting using security mode 2 (service level enforced security.)

Run-time errors:

Error code	Meaning
F0h	Mismatch parameter number
F1h	Mismatch parameter type

Function #15 `.fcBTHold` Control power-saving mode

Syntax: `CALL "BT.FN3" .fcBTHold INTERVAL%`

Description: This function shifts into the power-saving mode or disable it. For further details on the power-saving mode, refer to Section 18.2.4.4 "Power Supply Control."

Parameters: *`INTERVAL%`*

HOLD mode interval (0, 1 to 128 (unit: 100 ms))

Setting *`INTERVAL%`* to 0 disables the use of the power-saving mode.

Returned value: *(NONE)*

Note: The function should be executed after execution of `OPEN "COM4:M"` or `OPEN "COM4:S"` statement.

If the connected remote device does not support the HOLD mode, there is no transition, and the operation aborts with run-time error 640h.

Sending duplicate requests for shifts to the power-saving mode produces the run-time error 641h.

Run-time errors:

Error code	Meaning
05h	Parameter out of the range
34h	Bad file name or number. (The Bluetooth communications device file is not opened)
F0h	Mismatch parameter number
F1h	Mismatch parameter type
105h	Power-off detected.
640h	Failed to shift to power-saving mode
641h	Already in power-saving mode

Function #20 `.fcBTChkSnd`

Check data transmit result

Syntax: `CALL "BT.FN3" .fcBTChkSnd STATUS%`

Description: This function checks whether all transmit data has been transmitted to the remote device.

Parameters: *(None)*

Returned value: *STATUS%* Status

STATUS%	Current Connection Status
0	Transmission complete
1	Transmission not complete

Note: The function should be executed after execution of `OPEN "COM4:M"` or `OPEN "COM4:S"` statement.

Run-time errors:

Error code	Meaning
34h	Bad file name or number. (The Bluetooth communications device file is not opened)
F0h	Mismatch parameter number
F1h	Mismatch parameter type
105h	Power-off detected.

Example:

Transmit a message and wait for completion of the transmission

```
PRINT #4, "1234567890"              ' Transmit data
TIMEA = 50                          ' Transmission timeout: 5 seconds
SLOOP% = 1
WHILE TIMEA<>0 AND SLOOP% = 1       ' Wait for transmission completion
                                    ' or timeout
  CALL "BT.FN3" .fcBTChkSnd STATUS%
  IF STATUS% = 0 THEN
    SLOOP% = 0
  ENDIF
WEND
```

Function #21 `.fcBTGetSvc`
Read service discovery results

Syntax: `CALL "BT.FN3" .fcBTGetSvc NUM%,SCLASS%[()],SNAME$[()]`

Description: This function reads results from service discovery with an `OPEN "COM4:M,SDAP"` statement.

Parameters: *(None)*

Returned value: *NUM%* Number of services found

SCLASS%[()]

Service classes found

SCLASS%	Service Class
0	UNKNOWN_SERVICE_CLASSES
1	SERIAL_PORT
2	LAN_ACCESS_USING_PPP
3	DIALUP_NETWORKING
4	IRMC_SYNC
5	OBEX_OBJECT_PUSH
6	OBEX_FILE_TRANSFER
7	IRMC_SYNC_COMMAND
8	HEADSET
9	CORDLESS_TELEPHONY
10	INTERCOM
11	FAX
12	HEADSET_AUDIO_GATEWAY

SNAME$[()]

Service names found

NUM% gives the number of valid entries in the arrays *SCLASS%* and *SNAME$*.

If *NUM%* is greater than 1, treat *SCLASS%* and *SNAME$* as array variables.

If the number of services found exceeds the number of *SCLASS%* and

SNAME$ entries, the interface stops when the arrays are full.

If the service name is longer than the string length of *SNAME$*, the interface discards the excess bytes.

Note: The function should be executed after execution of `OPEN "COM4:M,SDAP"` statement.

Run-time errors:

Error code	Meaning
34h	Bad file name or number. (The Bluetooth communications device file is not opened)
F0h	Mismatch parameter number
F1h	Mismatch parameter type
105h	Power-off detected.

Example:

Query and browse for service and display the results.

```
DIM SCLASS%(4)                          ' Allocate space for 5 entries
DIM SNAME$(4)                           ' Allocate space for 5 entries
OPEN "COM4:M,SDAP,112233AABBCC" AS #4 ' Service discovery
CALL "BT.FN3" .fcBTGetSvc NUM%, SCLASS%(), SNAME$()
                                        ' Read search results
FOR I%=0 TO NUM%-1
   PRINT SCLASS%(I%);SNAME$(I%)         ' Display services found
NEXT
CLOSE #4                                ' Close Bluetooth communications
                                        ' device file
```

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 rou-tine.)
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 drive B Not allowed to access a read-only file
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	Out of string variable space Insufficient number of array variable elements

Error code	Meaning
100h	Cannot specify communications pathway
101h	Cannot connect to communications pathway
102h	Communications pathway not specified
103h	Communications pathway already connected
104h	Communications pathway already disconnected
105h	Power-off detected
106h	An internal error has occurred in the TCP/IP module during data transmission.
107h	The TCP/IP module has not been initiated.
108h	The memory for the TCP/IP module has became insufficient during data transmission.
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 disconnection.
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.

Error code	Meaning
293h	The problem occurred on the communication pathway.
295h	There is no user for login request.
600h	Failed to open a Bluetooth communications device file.
601h	Failed to connect.
602h	Connection timed out.
610h	Bluetooth data link already disconnected.
620h	Not connected to a remote device.
621h	Failed to get remote device name.
622h	No response from Bluetooth interface.
630h	No services found.
631h	Service discovery timed out.
640h	Failed to shift to power-saving mode.
641h	Already in power-saving mode

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 variable (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、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、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 RFG 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 types*] 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

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	[illegible]	␣	␣	0	@	P	`	p			␣	ー	タ	ミ	α	p
0001 1	[illegible]	␣	!	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	g	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	$^{-1}$	Ч
1010 A	␣	␣	*	:	J	Z	j	z			ェ	コ	ハ	レ	j	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 state-ment. (Refer to APLOAD statement in Chapter 14.)

NOTE 2: 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.
They can be switched to other national characters (see Appendix C2) by COUNTRY$ function. (Refer to COUNTRY$ function in Chapter 15.)

NOTE 3: BS (08h) is a backspace code.

NOTE 4: CR (0Dh) is a carriage return code.

NOTE 5: C (18h) 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.

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.

Appendix D
I/O Ports

■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	Monitors the following:			
`.pnEvent`	0	0	Keyboard buffer	–	0	No data
				`.pvEvKeyOn`	1	Data stored
		1	Barcode buffer	–	0	No data
				`.pvEvBarOn`	1	Data stored
		2	Trigger switch *1	–	0	OFF
				`.pvEvTrgOn`	1	ON
		3	Receive buffer	–	0	No data
				`.pvEvtCmOn`	1	Data stored
		4	Value of TIMEA function	–	0	Nonzero
				`.pvEvTma0`	1	Zero
		5	Value of TIMEB function	–	0	Nonzero
				`.pvEvTmb0`	1	Zero
		6	Value of TIMEC function	–	0	Nonzero
				`.pvEvTmc0`	1	Zero
		7	CS (CTS) signal *2	–	0	OFF or file closed
				`.pvEvCsOn`	1	ON
`.pnLCDCnt`	3	2-0	LCD contrast level *3	0 to 7 (0: Lowest, 7: Highest)		
`.pnMgLng`	4	0	Message version *4	`.pvSysMSG`	0	Japanese
				`.pvEnglish`	1	English

Port No.		Bit assign-ment	Monitors the following:			
.pnWupCtrl	8	0	Wakeup function	–	0	Deactivated
				.pvWupOn	1	Activated
		1	Initiation of BHT *5	–	0	Initiated by the power key
				.pvWupPwOn	1	Initiated by the wakeup function
		2	TIME$ function	–	0	System time selected
				.pvWupTmSt	1	Wakeup time selected
		3	Wakeup time	–	0	Not set
				.pvWupTmOn	1	Set
.pnSysSts	Eh	7-0	System status indication	.pvSysOff	0	OFF
				.pvsysOn	1	ON
.pnBarRrd	Fh	7-0	Re-read prevention enabled time *6	0-255		
–	10h-40Fh	7-0	VRAM *7	–	0	OFF
				–	1	ON
.pnBtVolt	6010 h	7-0	Battery voltage level *8	0-255		
.pnBtType	6011 h	0	Battery type	.pvBtRcrg	0	Rechargeable battery cartridge
				.pvBtDry	1	Dry cells

Port No.		Bit assignment	Monitors the following:			
.pnMKey	6040h	0	Magic key 1	–	0	Released
				.pvM1kyOn	1	Held down
		1	Magic key 2	–	0	Released
				.pvM2kyOn	1	Held down
		2	Magic key 3	–	0	Released
				.pvM3kyOn	1	Held down
		3	Magic key 4	–	0	Released
				.pvM4kyOn	1	Held down
.pnCmPrtcl	6060h	7-0	Communications protocol *9	.pvCPBHT	0	BHT-protocol
				.pvCPBHTIr	2	BHT-Ir protocol
.pnBHTIDL	6061h	7-0	ID (lower byte) *10	0-255		
.pnBHTIDH	6062h	7-0	ID (lower byte) *10	0-255		
.pnFont	6080h	0	Display font size	.pvFtStd	0	Standard-size
				.pvFtSmall	1	Small-size
.pnBprVib	6090h	0	Beeper	–	0	Deactivated
				.pvBprOn	1	Activated
		1	Vibrator	–	0	Deactivated
				.pvVibOn	1	Activated
.pnKeyEnt	60B0h	0	Key entry system	.pvKyNm	0	Numeric entry
				.pvKyAlpNm	1	Alphanumeric entry
.pnKeyMd	60B1h	0	Key entry mode	.pvKMNm	0	Numeric
				.pvKMAlp	1	Alphanumeric
.pnBprVolm	60C0h	1-0	Beeper volume *11	0-3		
.pnDfrgSzL	60E0h	7-0	Drive size to be defragmented (lower byte) *12	0-255		
.pnDfrgSzH	60E1h	7-0	Drive size to be defragmented (upper byte) *12	0-255		

Port No.		Bit assign-ment	Monitors the following:			
.pnRwuCtrl	60F0h	0	Remote wakeup function *13	.pvRwuOff	0	Deactivated
				.pvRwuOn	1	Activated
.pnRwuSpd	60F1h	2-0	Transmission speed for remote wakeup *14	.pvRwu96	001	9600bps
				.pvRwu192	010	19200bps
				.pvRwu384	011	38400bps
				.pvRwu576	100	57600bps
				.pvRwu1152	101	115200bps
.pnRwuHost	60F2h	0	Execution record of remote wakeup *15	.pvRwuRgst	1	Woken up remotely
		1	Termination of remote wakeup *16	.pvRwuEdOk	1	Terminated nor-mally
.pnRwuEfT	60F4h	7-0	Effective time for remote wakeup *17	1 to 24(hours)		

*1 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.

*2 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 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 In System Mode, the message version appears as English or Japanese on the LCD.

*5 If the BHT is initiated by the wakeup function, then this bit goes ON (1).

*6 The BHT 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 does not read same bar codes in succession.

*7 An 8-bit binary pattern (bits 7 to 0) on the input ports (which read VRAM) 10h to 1DBFh rep-resents a basic dot pattern column of the LCD. Bit value 1 means a black dot. The port number gives the dot column address.

[*8] 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 operation and it is not in proportion to the battery capacity, so use this voltage level as a reference value.

[*9] A user program returns the communications protocol type used for file transmission with the XFILE statement.

[*10] 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.

[*11] A user program returns the beeper volume level--01h (Low), 02h (Medium), or 03h (High). 00h means no beeping.

[*12] 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.

[*13] If "0" is returned, the remote wakeup function is deactivated; if "1," the function is activated.

[*14] The transmission speed to be applied when activating the remote wakeup will be returned.

[*15] If the BHT was woken up remotely at the last powering on, then "1" will be returned; if the BHT is initiated from any other means, "0" will be returned.

[*16] 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.

[*17] A user program returns the timeout period during which the BHT will be ready to receive remote wakeup commands from the host computer.

■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 assign-ment	Controls the following:			
`.pnLEDCtrl`	1	0	Indicator LED (red) *1	–	0	OFF
				`.pvLEDRed`	1	ON
		1	Indicator LED (green) *1	–	0	OFF
				`.pvLEDGrn`	1	ON
`.pnLCDCnt`	3	2-0	LCD contrast level *2	0 to 7 (0: Lowest, 7: Highest)		
`.pnMgLng`	4	0	Message version	`.pvSysMSG`	0	Japanese
				`.pvEnglish`	1	English
`.pnSlpTime`	6	7-0	Sleep timer *3	`0-255`		
`.pnWupCtrl`	8	0	Wakeup function *4	–	0	Deactivate
				`.pvWupOn`	1	Activate
		2	TIME$ function *5	–	0	Select the system time
				`.pvWupTmSt`	1	Select the wakeup time
`.pnSysSts`	Eh	0	System status indication	`.pvSysOff`	0	OFF
				`.pvsysOn`	1	ON
`.pnBarRrd`	Fh	7-0	Re-read prevention enabled time *6	`0-255`		
–	10h-40Fh	7-0	VRAM *7	–	0	OFF
				–	1	ON
`.pnSysMd`	6000h	0	Initiation of System Mode *8	`.pvSMdNGo`	0	Do not initiate
				`.pvSMdGo`	1	Initiate
`.pnBLCtrl`	6020h	0	Backlight *9	`.pvBLOff`	0	Turn OFF
				`.pvBLOn`	1	Turn ON
`.pnBLTime`	6021h	7-0	Backlight ON-duration *9	`0-255`		

<table>
<tr><th colspan="2">Port No.</th><th>Bit assign-ment</th><th colspan="4">Controls the following:</th></tr>
<tr><td>.pnTmPOff</td><td>6030h</td><td>7-0</td><td>Effective held-down time of power key *10</td><td>-</td><td colspan="2">1-255</td></tr>
<tr><td rowspan="2">.pnCmPrtcl</td><td rowspan="2">6060h</td><td rowspan="2">1-0</td><td rowspan="2">Communications protocol *11</td><td>.pvCPBHT</td><td>0</td><td>BHT-protocol</td></tr>
<tr><td>.pvCPBHTIr</td><td>2</td><td>BHT-Ir protocol</td></tr>
<tr><td>.pnBHTIDL</td><td>6061h</td><td>7-0</td><td>ID (lower byte) *12</td><td colspan="3">0-255</td></tr>
<tr><td>.pnBHTIDH</td><td>6062h</td><td>7-0</td><td>ID (upper byte) *12</td><td colspan="3">0-255</td></tr>
<tr><td rowspan="2">.pnFont</td><td rowspan="2">6080h</td><td rowspan="2">0</td><td rowspan="2">Display font size</td><td>.pvFtStd</td><td>0</td><td>Standard-size</td></tr>
<tr><td>.pvFtSmall</td><td>1</td><td>Small-size</td></tr>
<tr><td rowspan="4">.pnBprVib</td><td rowspan="4">6090h</td><td rowspan="2">0</td><td rowspan="2">Beeper *13</td><td>-</td><td>0</td><td>Deactivate</td></tr>
<tr><td>.pvBprOn</td><td>1</td><td>Activate</td></tr>
<tr><td rowspan="2">1</td><td rowspan="2">Vibrator *13</td><td>-</td><td>0</td><td>Deactivate</td></tr>
<tr><td>.pvVibOn</td><td>1</td><td>Activate</td></tr>
<tr><td rowspan="2">.pnKeyEnt</td><td rowspan="2">60B0h</td><td rowspan="2">0</td><td rowspan="2">Key entry system</td><td>.pvKyNm</td><td>0</td><td>Numeric entry</td></tr>
<tr><td>.pvKyAlpNm</td><td>1</td><td>Alphanumeric entry</td></tr>
<tr><td rowspan="2">.pnKeyMd</td><td rowspan="2">60B1h</td><td rowspan="2">0</td><td rowspan="2">Key entry mode</td><td>.pvKMNm</td><td>0</td><td>Numeric</td></tr>
<tr><td>.pvKMAlp</td><td>1</td><td>Alphabet</td></tr>
<tr><td>.pnBprVolm</td><td>60C0h</td><td>1-0</td><td>Beeper volume *14</td><td colspan="3">0-3</td></tr>
<tr><td>.pnDfrgSzL</td><td>60E0h</td><td>7-0</td><td>Drive size to be defragmented (lower byte) *15</td><td colspan="3">0-255</td></tr>
<tr><td>.pnDfrgSzH</td><td>60E1h</td><td>7-0</td><td>Drive size to be defragmented (upper byte) *15</td><td colspan="3">0-255</td></tr>
<tr><td rowspan="3">.pnDfrgGo</td><td rowspan="3">60E2h</td><td rowspan="3">1-0</td><td rowspan="3">Execution of defragmentation *16</td><td>.pvDFNoDsp</td><td>0</td><td>Defragment w/o bar graph</td></tr>
<tr><td>.pvDFAGrph</td><td>1</td><td>Defragment w/ absolute bar graph</td></tr>
<tr><td>.pvDFRGrph</td><td>2</td><td>Defragment w/ relative bar graph</td></tr>
<tr><td rowspan="2">.pnRwuCtrl</td><td rowspan="2">60F0h</td><td rowspan="2">0</td><td rowspan="2">Remote wakeup function *17</td><td>.pvRwuOff</td><td>0</td><td>Deactivate</td></tr>
<tr><td>.pvRwuOn</td><td>1</td><td>Activate</td></tr>
<tr><td rowspan="5">.pnRwuSpd</td><td rowspan="5">60F1h</td><td rowspan="5">2-0</td><td rowspan="5">Transmission speed for remote wakeup *18</td><td>.pvRwu96</td><td>001</td><td>9600bps</td></tr>
<tr><td>.pvRwu192</td><td>010</td><td>19200bps</td></tr>
<tr><td>.pvRwu384</td><td>011</td><td>38400bps</td></tr>
<tr><td>.pvRwu576</td><td>100</td><td>57600bps</td></tr>
<tr><td>.pvRwu1152</td><td>101</td><td>115200bps</td></tr>
<tr><td>.pnRwuEfT</td><td>60F4h</td><td>7-0</td><td>Effective time for remote wakeup *19</td><td colspan="3">1 to 24(hours)</td></tr>
</table>

[*1] The indicator 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 indicator LED to OFF in the `OPEN "BAR:"` statement, then a user program can control the indicator LED although the bar code device file is opened.

[*2] 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
```

[*3] 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
```

[*4] To activate the wakeup function, set 1 to this bit; to deactivate it, set 0.

[*5] 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.

[*6] 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 does not read same bar codes in succession. The default is 10 (1 second).

[*7] An 8-bit binary pattern (bits 7 to 0) on the output ports (which are stored in the VRAM) 10h to 1DBFh represents a basic dot pattern column of the LCD. Bit value 1 means a black dot.

The port number gives the dot column address.

[*8] Refer to Appendix H, "Program file named APLINT.PD3."

[*9] 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.

[*10] You can set the held-down time of the power key required for powering off the BHT. The setting range is from 0.1 to 25.5 seconds in increments of 0.1 second. The default is 5 (0.5 second).

[*11] You can set the communications protocol type for transmitting files with the XFILE statement.

[*12] 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, for example, 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
```

[*13] If you set 0 (Deactivate) to both bits 0 and 1, only the beeper will work.

[*14] The beeper volume level may be adjusted to four levels--01h (Low), 02h (Medium), 03h (High), and 00h (OFF).

[*15] 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, for example, 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.

[*16] 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 defrag-mentation, 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 absolute bar
                'graph
```

[*17] To activate the remote wakeup, set "1"; to deactivate, set "0."

[*18] Set the transmission speed to be applied for remote wakeup.

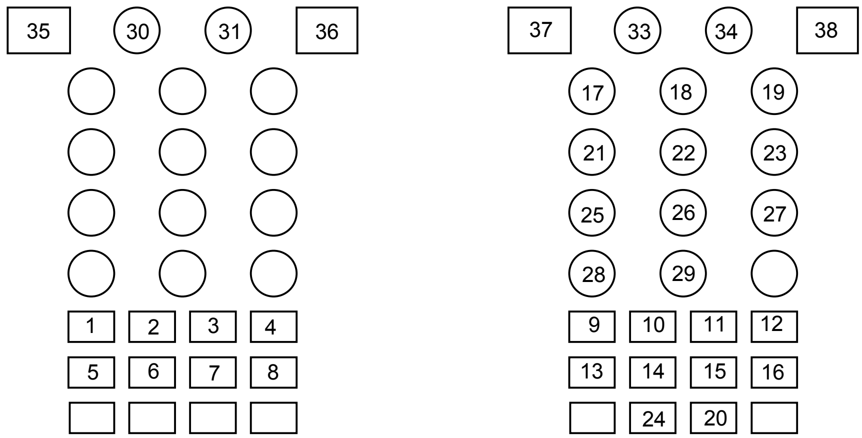
[*19] You may set the timeout period during which the BHT will be ready to receive remote wakeup commands from the host computer.

Appendix E
Key Number Assignment on the Keyboard

■Key Number Assignment

The keys on the BHT keyboard are assigned numbers as shown below.

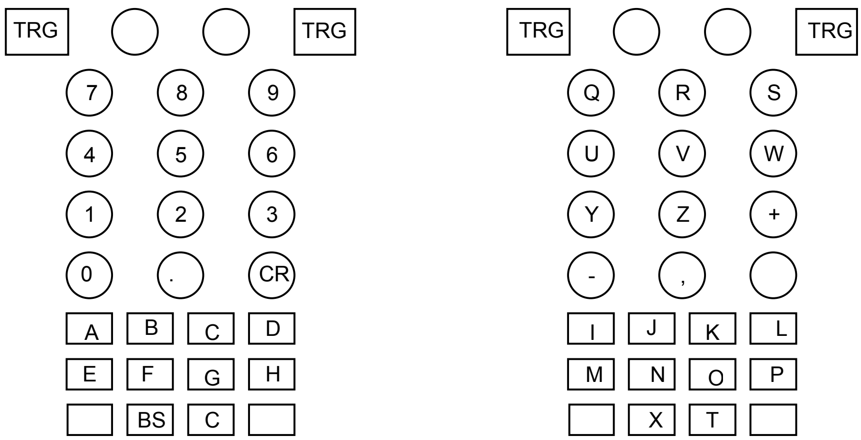
Non-shift mode　　　　　　　　Shift mode



■Default Data Assignment

The default data assignment is shown below.

Non-shift mode　　　　　　　　Shift mode



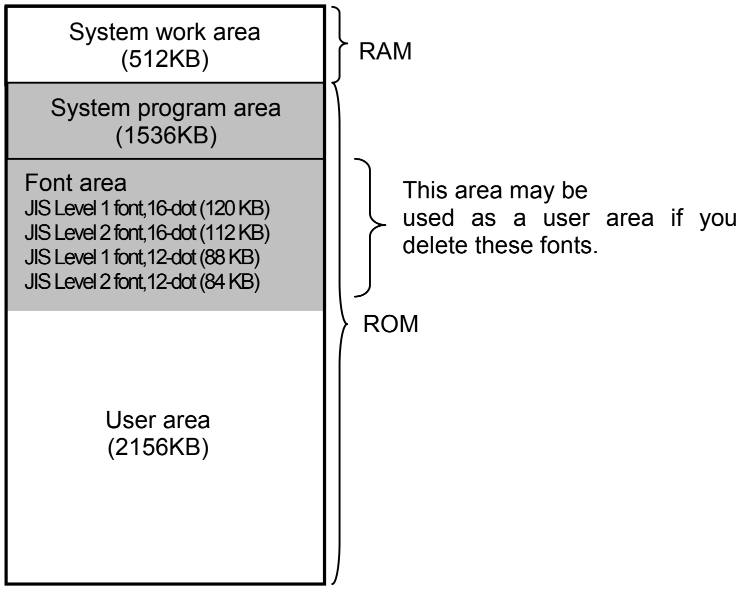
BS, CR, and C are a backspace (08h), carriage return (0Dh), and cancel (18h) code, respectively.

Appendix F
Memory Area

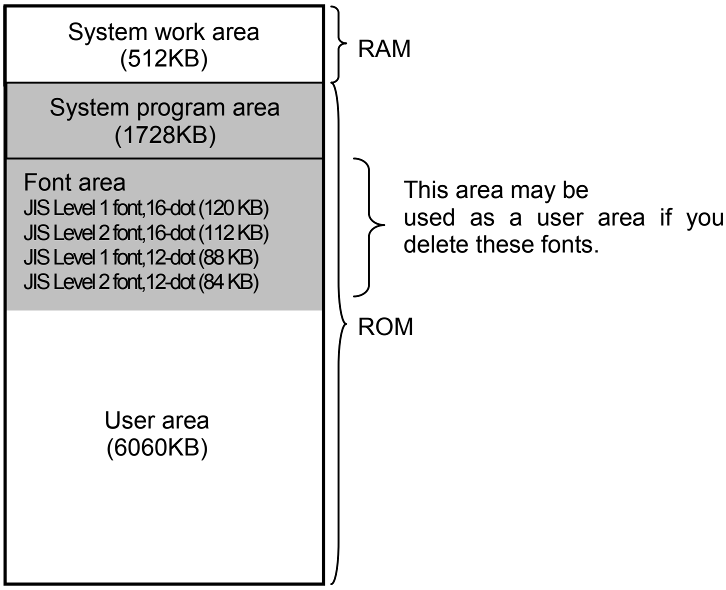
■**Memory Map**

The memory maps are shown below.

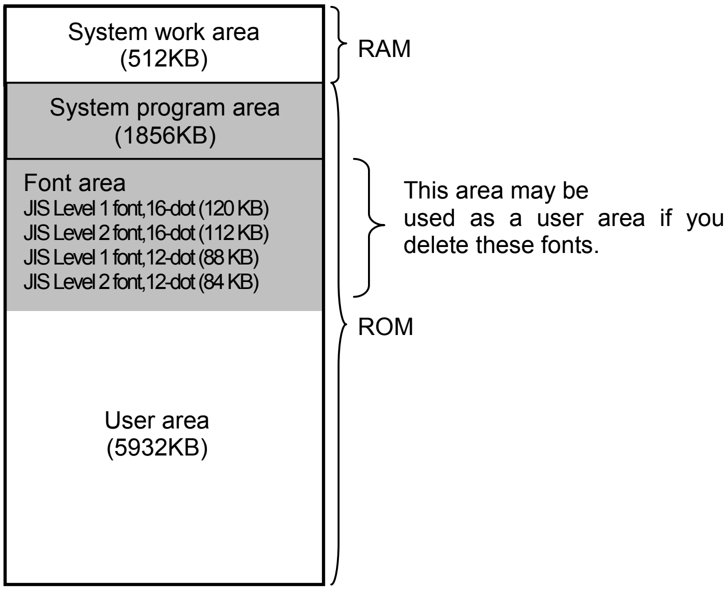
ROM 4MB, RAM 512KB Type



ROM 8MB, RAM 512KB Type (except BHT8048DB)



BHT8048DB



■**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.

The maximum allowable size for a single user program is 64 kilobytes excluding register variables.

■Battery Backup of Memory

The BHT backs up the memory with a battery cartridge. Therefore, data stored in the memory 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	32KB
Register variable area	64KB

Each variable occupies the memory space as listed below.

Variables	Max. memory space
Integer variable	2 bytes
Real variable	6 bytes
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).

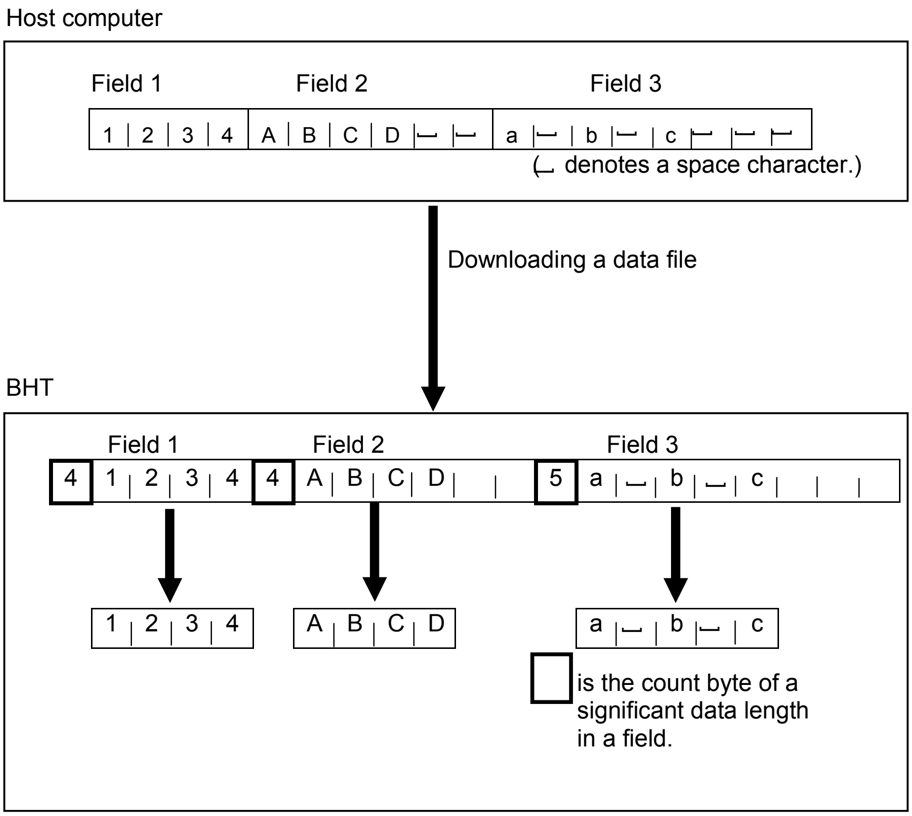
Appendix G
Handling Space Characters in Downloading

■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 trailing spaces padded in the tail of each data field.

The BHT has a new feature which can handle trailing spaces in a data field as data.

The figure below shows the process in which the spaces used as padding characters are eliminated. (Note that spaces between a and b and between b and c in field 3 are not padding characters.)



■**To handle space characters as data**

To handle trailing spaces in 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 spaces in its tail by using a `SEARCH` function, for instance, use any of the following methods:

<u>Example 1</u> After downloading a data file, fill the unused spaces in each field with spaces
and then search for the target field data.

A	B	C	␣	␣	Send data
A	B	C			Receive data
A	B	C	␣	␣	Filling with space characters
A	B	C	␣	␣	Search data to be specified (␣ denotes a space character.)

<u>Example 2</u> Before downloading a data file, substitute any of the characters which will not be
used as effective data, e.g., an asterisk (*), for the spaces in the host computer.

A	B	C	*	*	Send data
A	B	C	*	*	Receive data
A	B	C	␣	␣	Data to be searched
A	B	C	*	*	Search data to be specified (␣ denotes a space character.)

<u>Example 3</u> When specifying a field data to be searched, do not include trailing spaces in a data field.

A	B	C	␣	␣

Send data

A	B	C		

Receive data

A	B	C	␣	␣

Data to be searched

A	B	C

Search data to be specified
(␣ denotes a space character.)

■**To make the BHT handle space characters as data**

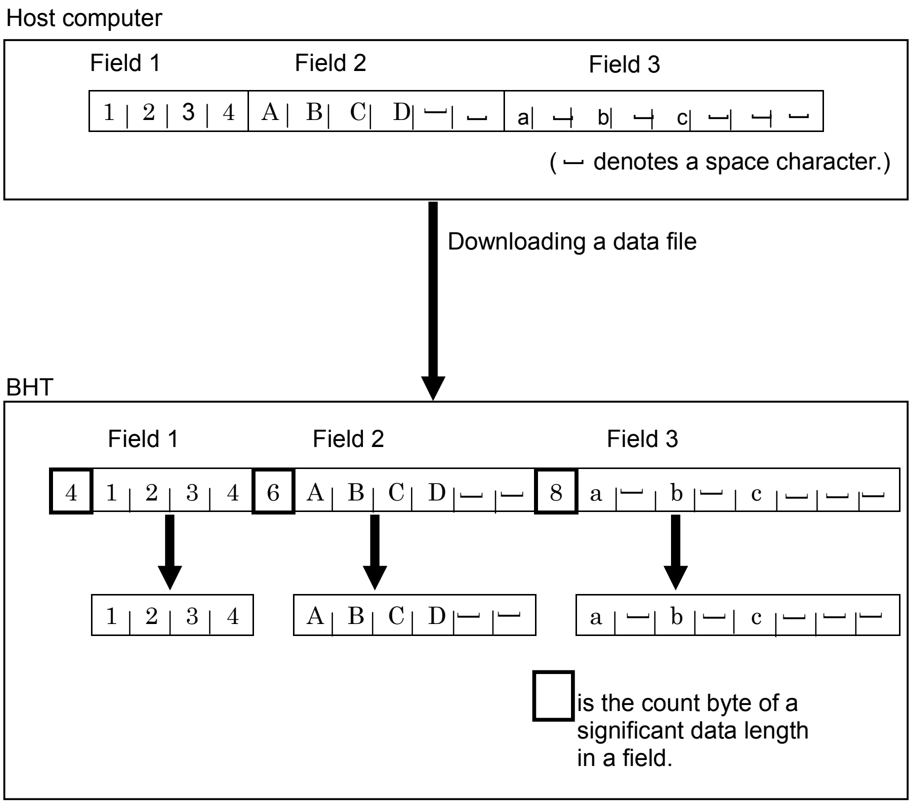
You can specify the handling of trailing spaces in a data field with System Mode or an `XFILE` statement.

System Mode: To handle trailing spaces as data, select "Data" in FIELD SPACE item on the SET PROTOCOL screen of the SET SYSTEM menu.

`XFILE` statement: To handle trailing spaces as data, specify `T` to `"protocolspec"` in the `XFILE` statement.

```
XFILE "d2.dat","T"
```

The figure below shows the process in which trailing spaces in a data field are handled as data in the BHT.



Appendix H
Programming Notes

■Program file named APLINT.PD3

If a program file named APLINT.PD3 is stored in the BHT, 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: Not start System Mode (default) 1: Start 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 BS key or C 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 entry.
      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-han-dling 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-defined 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 (as local variables.)
`READ`	Reads data defined by DATA statement(s) and assigns them to variables.
`RESTORE`	Specifi es 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 a backlight function on/off key. This statement also defines a magic key as a 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.
`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 display mode (screen mode, and font size) and character attributes (character enlargement, and font reverse attributes).
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 a backlight function on/off key. This statement also defines a magic key as a 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.

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 speci-fied 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. This statement also activates or deactivates the indicator LED and the beeper (vibrator) individually.
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 the bar code type and the number of digits of a 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 data 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 the value of a numeric expression into a string.
`VAL`	Converts a string into a numeric value.

Creating user-defined functions

Statements	
`DEF FN`	Names and defines a user-defined function.
`DEF FN...END DEF`	Names and defines a user-defined function.
`FUNCTION...END FUNCTION`	Names and defines user-defined function FUNCTION.
`SUB...END SUB`	Names and defines user-defined 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	

Supplement

CONTENTS

Suuplement A What's different from the BHT-5000?

A.1 Communication

Item		BHT-5000	BHT-8000
Optical interface	Communications operation	Full duplex	Half duplex
	Transmission speed	300, 600, 1200, 2400, 4800, 9600, 19200, 38400 bps	2400, 9600, 19200, 38400, 57600, 115200 bps
	Parity bit	None, Odd, or Even	None
	Character length	7 or 8 bits	8 bits
	Stop bits	1 or 2 bits	1 bit
	Signal lines	SD, RD, RS, CS	SD, RD
Direct-connect interface	Transmission speed	300, 600, 1200, 2400, 4800, 9600, 19200, 38400 bps	300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bps
Simultaneous opening with the bar code device file		Possible	Optical (IrDA) interface: Not possible Direct-connect interface: Possible
Communications protocol		BHT-protocol Multilink protocol	BHT-protocol BHT-Ir protocol
File transmission with XFILE statement	Receive file with the name given by the sender	Not possible	Possible
	Receive file with different name	Not possible	Possible

- **Setting the transmission speed for IrDA communication**

  For communication between the BHT-8000 and the host via the CU, you need to set the transmission speed of the CU to the same value as that of the BHT using the DIP switch located at the bottom of the CU.

- **Switching time between sending and receiving on the IrDA interface**

  For IrDA communication with the BHT-8000, the IrDA interface should satisfy the following requirements in switching between sending and receiving:

  a) Within 10 ms from completion of sending, the IrDA interface should become ready to receive.

  b) After 10 ms or more from completion of receiving, the IrDA interface should start sending.

- **Note for specifying communications parameters for the IrDA interface**

  If you specify communications parameters not supported by the IrDA interface in the BHT-8000, the following will result.

Communications parameters	Parameters not supported by IrDA interface	Execution result
Transmission speed	300, 600, 1200, 4800 bps	Run-time error
Parity bits	Odd or Even	None
Character length	7 bits	8 bits
Stop bit(s)	2 bits	1 bit

A.2 Bar code reading

Item	BHT-5000	BHT-8000
OPEN "BAR:" statement extension*1	Not available	Available
Code 128 special characters conversion*[1]	Conforms to the 1986 USS Standard	Conforms to the 1993 USS Standard (Conversion system differs in some parts)
Length of beep at completion of reading*[1]	100 ms	60 ms
Drive vibrator at completion of reading*[1]	Not available	Available
Bar code types that can be specified by CHKDGT$ function*[2]	EAN-13, EAN-8, UPC-A, UPC-E, ITF, Code 39, Codabar (NW-7)	EAN-13, EAN-8, UPC-A, UPC-E, STF, ITF, Code 39, Codabar (NW-7)
If bar code data contains characters out of the specification, CHKDGT$ function returns: *[2]	Calculation result	Null string

*[1] For details, refer to OPEN "BAR:" in Chapter 14 "Statement Reference."

*[2] For details, refer to CHKDGT$ in Chapter 15 "Function Reference."

A.3 Screen display

Item		BHT-5000	BHT-8000
BHT-2000 compatible mode		Available	Not available
Chars x Lines (Dots, W x H)	ANK*[1]	21 x 8 (6 x 8)	Standard-size font:21 x 8 (6 x 8)
			Small-size font: *[2] 21 x 10 (6 x 6)
	Kanji	Full-width: 8 x 4 (16 x 16)	Standard-size font Full-width: 8 x 4 (16 x 16) Half-width: 16 x 4 (8 x 16)
		Half-width: 16 x 4 (8 x 16)	Small-size font *[2] Full-width: 10 x 5 (12 x 12) Half-width: 21 x 5 (6 x 12)
	Condensed Kanji	Full-width: 10 x 4 (12 x 16)	Not available *[3]
		Half-width: 21 x 4 (6 x 16)	
Double-width		Not available	Available
No. of user-defined fonts loadable		ANK: 32 fonts Kanji: 32 fonts	ANK: 32 fonts Kanji: 128 fonts
User-defined cursor load/display function *[4]		Not available	Available
Characters that COUNTRY$ function can display		ANK only	ANK and half-width Kanji

*[1] ANK: Alphanumerics and Katakana

*[2] Switching between the standard-size and small-size fonts may be specified by the OUT statement. For the setting procedure, refer to Chapter 7 "I/O Facilities, "Chapter 14 "Statement Reference, OUT," and Appendix D "I/O Ports."

*[3] In the BHT-8000, specifying the condensed Kanji mode will result in a run-time error.

*[4] This function displays a cursor in the shape defined by the user. The cursor shape may be defined with the APLOAD or KPLOAD statement. The defined cursor may be displayed with the LOCATE statement. Refer to Chapter 14 "Statement Reference."

A.4 Keyboard

Item	BHT-5000	BHT-8000
Magic keys *[1]	M1 and M2 keys	M1, M2, M3, and M4 keys
Default trigger switch	Dedicated trigger switch	M3 (left-hand) and M4 (right-hand) keys
Key number assignment range	1 to 34	1 to 31 and 33 to 38 (32 ignored)
Alphabet entry *[2]	Available in 32-keypad models only	Available (Alphabet entry mode added)

*[1] For definition of magic keys, refer to the KEY statement in Chapter 14 "Statement Reference."

*[2] For details about the alphabet entry, refer to Chapter 7, Section 7.2 "Input from the Keyboard."

A.5 Backlight

Item	BHT-5000	BHT-8000
Default backlight on/off control key	Trigger switch with SF key held down	M1 key with SF key held down
Key assignment numbers for backlight on/off control	0 to 34	0 to 38

A.6 Files

Item	BHT-5000	BHT-8000
File storage device(s)	RAM (Drive A) Flash ROM (Drive B)	Flash ROM (The RAM is used to run programs efficiently.)
Max. number of files loadable	RAM (Drive A): 40 Flash ROM (Drive B): 40	80
Cluster size	4 KB 8 KB (BHT-5079 only)	4 KB
User area	RAM (Drive A) 92 KB (BHT-5071) 464 KB (BHT-5075) 964 KB (BHT-5077) 1976 KB (BHT-5079) Flash memory (Drive B) 124 KB (380 KB*)	Max. 2156 KB (2560 KB *)
Defragment the drive	Not available	Available (Can be initiated by the user or automatically during auto power-off)
Specify drive B with FRE function	Available	Not available (Resulting in a run-time error)

* Values in parentheses are user areas available when font files are deleted.

- **Defragment the drive**

  To use the user area efficiently, the BHT-8000 supports the defragmentation of drive that can be initiated by the user or automatically. For details, refer to Chapter 8 "Files," Subsection 8.2.5 "Programming for Data Files."

- **Specify drive**

  In the BHT-8000, drive B is provided for ensuring compatibility with other BHT series. For details, refer to Chapter 8 "Files," Subsection 8.2.6 "About Drives."

A.7 Work and common variables

Item	BHT-5000	BHT-8000
Max. memory spaces available for work and common variables	6 KB	32 KB

A.8 Beeper & vibrator control

Item	BHT-5000	BHT-8000
Beeper volume adjustment in user programs	Not available	Available
Beeper and vibrator switching & control in user programs	Not available	Available
Drive the vibrator with BEEP statement	Not available	Available
Frequencies by the special beeper effects in BEEP statement	Low:1015 Hz Medium:2042 Hz High:4200 Hz	Low: 698 Hz Medium:1396 Hz High:2793 Hz
Frequency range that drives no beeper in BEEP statement	3 to 260 Hz	3 to 61 Hz

A.9 Sleep function

Item	BHT-5000	BHT-8000
Activate the sleep function when the sleep timer is set to 10 seconds or more in TIMEA/TIMEB/TIMEC function	No	Yes

A.10 Extended functions and exnsion library

Item	BHT-5000	BHT-8000
Extended functions	None	SYSTEM.FN3 (Read or write system settings from/to the memory) SYSMDFY.FN3 (Control system files) CRC.FN3 (Calculate a CRC)
Extension library	Exclusively designed.	Exclusively designed.

A.11 Remote wakeup

Item	BHT-5000	BHT-8000
Remote wakeup	Not available	Available

- **Remote wakeup**

  The remote wakeup function allows you to automatically wake up the BHT-8000 placed on the CU from a remote location by sending the specified command from the host computer to the BHT-8000. For details, refer to Chapter 12 "Power-related Functions," Section 12.4 "Remote Wakeup Function."

Supplement B What's different from the BHT-6000?

B.1 Communication

Item		BHT-6000	BHT-8000
File transmission with XFILE statement	Receive file with the name given by the sender	Not possible	Possible
	Receive file with different name	Not possible	Possible
Specify the output pulse width of IR beam		Possible	Not possible *

* Ignored if specified by the OUT statement.

B.2 Bar code reading

Item	BHT-6000	BHT-8000
OPEN "BAR:" statement extension*1	Not available	Available
Code 128 special characters conversion*1	FNC characters ignored	Conforms to the 1993 USS Standard (Conversion system differs in some parts)
Length of beep at completion of reading*1	100 ms	60 ms
Drive vibrator at completion of reading*1	Not available	Available
Bar code types that can be specified by CHKDGT$ function*2	EAN-13, EAN-8, UPC-A, UPC-E, ITF, Code 39, Codabar (NW-7)	EAN-13, EAN-8, UPC-A, UPC-E, STF, ITF, Code 39, Codabar (NW-7)
If bar code data contains characters out of the specification, CHKDGT$ function returns: *2	Calculation result	Null string

*1 For details, refer to OPEN "BAR:" in Chapter 14 "Statement Reference."

*2 For details, refer to CHKDGT$ in Chapter 15 "Function Reference."

B.3 Screen display

Item		BHT-6000		BHT-8000	
Chars x Lines (Dots, W x H)	ANK*1	Standard-size font: 16 x 6 (6 x 8)		Standard-size font: 21 x 8 (6 x 8)	
		Small-size font: 16 x 8 (6 x 6)		Small-size font: 21 x 10 (6 x 6)	
	Kanji	Standard-size font Full-width: 6 x 3 (16 x 16) Half-width: 12 x 3 (8 x 16)		Standard-size font Full-width: 8 x 4 (16 x 16) Half-width: 16 x 4 (8 x 16)	
		Small-size font Full-width: 8 x 4 (12 x 12) Half-width: 16 x 4 (6 x 12)		Small-size font Full-width: 10 x 5 (12 x 12) Half-width: 21 x 5 (6 x 12)	
Double-width		Not available		Available	
No. of user-defined fonts loadable		ANK: 32 fonts Kanji: 32 fonts		ANK: 32 fonts Kanji: 128 fonts	
User-defined cursor load/display function *2		Not available		Available	
Characters that COUNTRY$ function can display		ANK only		ANK and half-width Kanji	
Specification range in LOCATE and Returned value of POS/CLRLIN functions POS: Same as Row CLRLIN: Same as Column	ANK*1	Column	1 to 17	Column	1 to 22
		Row	Standard-size font: 1 to 6 Small-size font: 1 to 8	Row	Standard-size font: 1 to 8 Small-size font: 1 to 10
	Kanji	Column	Standard-size font: 1 to 13 Small-size font: 1 to 17	Column	Standard-size font: 1 to 17 Small-size font: 1 to 22
		Row	Standard-size font: 1 to 5 Small-size font: 1 to 7	Row	Standard-size font: 1 to 7 Small-size font: 1 to 9
VRAM size		576 bytes		1024 bytes	

*1 ANK: Alphanumerics and Katakana

*2 This function displays a cursor in the shape defined by the user. The cursor shape may be defined with the APLOAD or KPLOAD statement. The defined cursor may be displayed with the LOCATE statement. Refer to Chapter 14 "Statement Reference."

B.4 Keyboard

Item	BHT-6000	BHT-8000
Magic keys *[1]	M1 and M2 keys	M1, M2, M3, and M4 keys
Default trigger switch	M1 and M2 keys	M3 (left-hand) and M4 (right-hand) keys
Key number assignment range	1 to 31, 33, and 34	1 to 31 and 33 to 38 (32 ignored)
Alphabet entry *[2]	Available (entry procedure exclusively designed)	Available (entry procedure exclusively designed)

*[1] For definition of magic keys, refer to the KEY statement in Chapter 14 "Statement Reference."

*[2] For details about the alphabet entry, refer to Chapter 7, Section 7.2 "Input from the Keyboard."

B.5 Backlight

Item	BHT-6000	BHT-8000
Key assignment numbers for backlight on/off control	1-31, 33, and 34	0 to 38

B.6 Files

Item	BHT-6000	BHT-8000
File storage device(s)	RAM (Drive A) Flash ROM (Drive B)	Flash ROM (The RAM is used to run programs efficiently.)
Max. number of files loadable	RAM (Drive A): 40 Flash ROM (Drive B): 40	80
Cluster size	4 KB 8 KB (BHT-6049 only)	4 KB
User area	RAM (Drive A) 468 KB Flash memory (Drive B) 64 KB (BHT-6045) 568 KB (BHT-6047) 1584 KB (BHT-6049) (If font files are deleted) 320 KB (BHT-6045) 828 KB (BHT-6047) 1840 KB (BHT-6049)	Max. 2156 KB (2560 KB if font files are deleted)
Defragment the drive	Not available	Available (Can be initiated by the user or automatically during auto power-off)
Specify drive B with FRE function	Available	Not available (Resulting in a run-time error)

- **Defragment the drive**

  To use the user area efficiently, the BHT-8000 supports the defragmentation of drive that can be initiated by the user or automatically. For details, refer to Chapter 8 "Files," Subsection 8.2.5 "Programming for Data Files."

- **Specify drive**

  In the BHT-8000, drive B is provided for ensuring compatibility with other BHT series. For details, refer to Chapter 8 "Files," Subsection 8.2.6 "About Drives."

B.7 Work and common variables

Item	BHT-6000	BHT-8000
Max. memory spaces available fro work and common variables	6 KB	32 KB

B.8 Beeper & vibrator control

Item	BHT-6000	BHT-8000
Beeper volume adjustment in user programs	Not available	Available
Beeper and vibrator switching & control in user programs	Not available	Available
Drive the vibrator with BEEP statement	Not available	Available
Frequencies by the special beeper effects in BEEP statement	Low: 1033 Hz Medium: 2168 Hz High: 4337 Hz	Low: 698 Hz Medium: 1396 Hz High: 2793 Hz

B.9 Extended functions and extension library

Item	BHT-6000	BHT-8000
Extended functions	None	SYSTEM.FN3 (Read or write system settings from/to the memory) SYSMDFY.FN3 (Control system files) CRC.FN3 (Calculate a CRC)
Extension library	Exclusively designed	Exclusively designed

B.10 Remote wakeup

Item	BHT-6000	BHT-8000
Remote wakeup	Not available	Available

- **Remote wakeup**

  The remote wakeup function allows you to automatically wake up the BHT-8000 placed on the CU from a remote location by sending the specified command from the host computer to the BHT-8000. For details, refer to Chapter 12 "Power-related Functions," Section 12.4 "Remote Wakeup Function."

Supplement C
What's different from the BHT-7000?

C.1 Files

Item	BHT-7000	BHT-8000
Drive defragmentation will be initiated:	- Specified by the user in the OUT statement. - When updating or adding files is performed with insufficient free space of the user area.	- Specified by the user in the OUT statement. - When updating or adding files is performed with insufficient free space of the user area. - When the auto power-off function turns off the BHT.

- **Defragment the drive**

  For details, refer to Chapter 8 "Files," Subsection 8.2.5 "Programming for Data Files."

C.2 Battery voltage display key

Item	BHT-7000	BHT-8000
Define function key or magic key as battery voltage display key	Not possible	Possible*

* The BHT-8000 may define a magic key as a battery voltage display key. For details, refer to the KEY statement in Chapter 14 "Statement Reference."

C.3 Monitor the CU state

Item	BHT-7000	BHT-8000
Get the CU state	Can get the following three states: - CU with BHT placed - CU without BHT - CU loaded with dry battery cartridge	Can get the following state only: - CU without BHT

C.4 Scanning range marker

Item	BHT-7000	BHT-8000
Scanning range marker control in user programs	Possible	Not possible (No scanning range marker is provided.)

C.5 System status indicator

Item	BHT-7000	BHT-8000
System status indicator on/off control in user programs	Not possible (Always displayed)	Possible

C.6 Beeper

Item	BHT-7000	BHT-8000
Frequency range that drives no beeper in BEEP statement	3 to 39 Hz	3 to 61 Hz

C.7 Remote wakeup

Item	BHT-7000	BHT-8000
When the rechargeable battery cartridge is loaded	Available	Available
When the dry cells are loaded	Not available	Available
Effective period in which the BHT is ready to receive remote wakeup commands	When the BHT is placed on the CU after turned off.	Within the specified period after the BHT is turned off
Initiation of remote wakeup	WAKE command following specified commands sent from the host	Any message being sent from the host in succession for at least one minute at 30-ms intervals.

- **Remote wakeup**

  For details, refer to Chapter 12 "Power-related Functions," Section 12.4 "Remote Wakeup Function."

C.8 Key data assigned in the alphabet entry mode for the alphanumeric system

Key data assigned to the following three keys is different from that assigned in the BHT-7000:

Keys	BHT-7000	BHT-8000
3 key	Y, Z, +, y, z	Y, Z, space, y, z
0 key	-, %, $, \	+, -, *, \
Period(.) key	comma (,), /, space	/, $, %, comma (,)

Index

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

BHT-BASIC

(BHT-8000 series)

Programmer's Manual
3rd Edition, June 2003
DENSO WAVE INCORPORATED