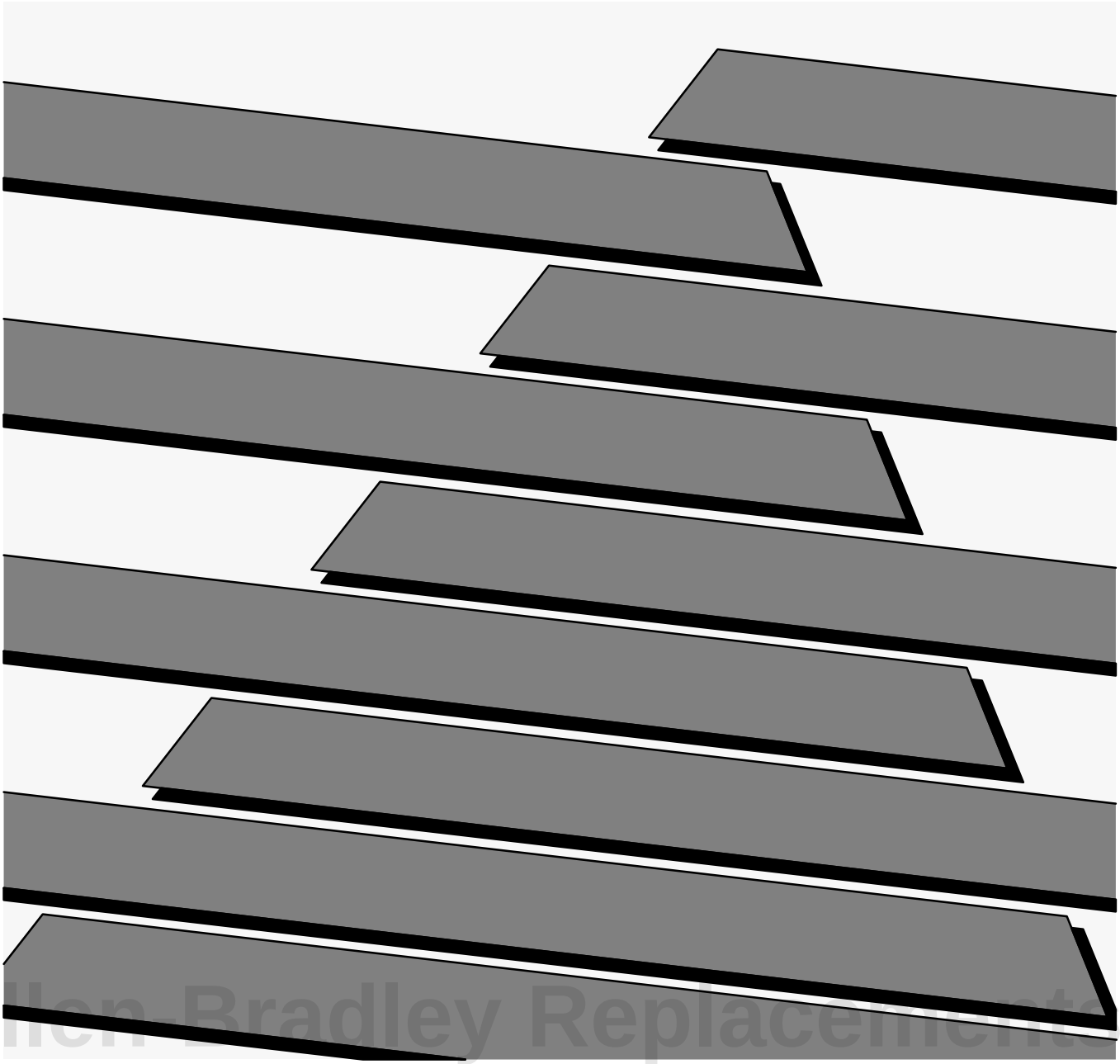
ALLEN-BRADLEY

Bulletin 2708
BASIC Language Development Kit

(Catalog No. 2708-NBD)

User Manual



Important User Information

Solid state equipment has operational characteristics differing from those of electromechanical equipment. “Application Considerations for Solid State Controls” (Publication SGI-1.1) describes some important differences between solid state equipment and hard–wired electromechanical devices. Because of this difference, and also because of the wide variety of uses for solid state equipment, all persons responsible for applying this equipment must satisfy themselves that each intended application of this equipment is acceptable.

In no event will the Allen-Bradley Company be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, the Allen-Bradley Company cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Allen-Bradley Company with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of the Allen-Bradley Company is prohibited.

Chapter 1

Introduction to BASIC

The BASIC Language Development Kit (Catalog No. 2708-NBD) software consists of BASIC source files which you may use to decrease development time. This kit contains:

- Software Diskette
- BASIC Language Development Kit User's Manual (Catalog No. 2708-ND004)
- Learning to Use Microsoft QuickBASIC – by Microsoft
- Programming in BASIC – by Microsoft
- BASIC Language Reference – by Microsoft

Note: The contents of this user's manual is also contained in the Bulletin 2708 Attended Workstation User's Manual (Catalog No. 2708-ND001).

What is BASIC?

The acronym "BASIC" stands for "Beginner's All-purpose Symbolic Instruction Code". It was designed as a programming language for novice programmers in the early 60's. In the 90's, BASIC still ranks as one of the most popular programming languages.

BASIC is used as a workstation programming language for several reasons:

1. its ease of use
2. its wide-spread popularity
3. its structure and versatility

What is QuickBASIC ?

Almost no one is using the original ANSI standard BASIC any more. The language only allowed 26 variables, had no string capability, and was very simplistic. QuickBASIC is a product of Microsoft Corporation which expands greatly on the original language, and provides an integrated development environment containing a BASIC program editor, a BASIC compiler with a vast array of statements, and a debugging platform.

The QuickBASIC program editor checks each line of the program for syntactical errors as they are entered. It has all the functions normally found in program editors and is designed to interact with the compiler and debugger. It facilitates the separation of functions and subroutines into easy-to-read pages, and has full search and replace facilities.

The QuickBASIC program compiler can be invoked from inside the editor, or from the DOS command line. Once an error is found, the compiler returns control to the editor on the line with the mistake so that changes can be made to correct the problem. When the program is compiled successfully, it can be run either from inside the debugger, or from the DOS command line. If a run-time error is encountered, the screen returns to the editor at the line containing the error.

In the event the program fails to do what was planned, the program debugger can be used to track down any logical problems.

These tools are essential for programmers at any level. We highly recommend that you make use of Microsoft's QuickBASIC to develop application programs for the workstations.

What is A-B BASIC?

A-B BASIC is a subset of Microsoft's QuickBASIC, consisting of those commands which are relevant in the workstation environment. A-B BASIC includes a set of Special Devices which allows access to things like bar code scanners. Many commands which would be appropriate in a PC environment, but are not applicable on a workstation, such as color, graphics, and disk commands have been left out.

A-B BASIC provides the user with a near-industry-standard programming language which is powerful enough to handle any application, yet easy to learn.

Additional reference resources for learning about QuickBASIC and A-B BASIC are:

Learning to Use Microsoft QuickBASIC – by Microsoft
Programming in BASIC – by Microsoft
Basic Language Reference – by Microsoft

Note: These publications are provided with the Application Generator Software (Catalog No. 2708-NAG) and the BASIC Language Development Kit (Catalog No. 2708-NBD).

Together, they provide all the information needed by a programmer with some BASIC experience. However, none of the books can be considered a *tutorial* on the BASIC language. Check your local library, the computer section of your local bookstore or a computer dealer near you for beginner BASIC books. Most programmers have programmed in BASIC or a similar language and will have little trouble converting to A-B BASIC.

Chapter 2

Developing and Running an A-B BASIC Program

The A-B BASIC Development Procedure

Step #1 – Planning

The 2708-DH5AXX workstations have approximately 32K bytes and the 2708-DH5BXX workstations have approximately 79K bytes dedicated for program memory. Programs that are larger than about 20-30 pages of BASIC source code need to be broken up into several smaller CHAINed programs. It is much easier to plan for this in the design stage than to be forced into it in the coding stage.

For smaller applications (applications that can be coded in 20-30 pages of source or less) there is no need to worry about the CHAIN statement. By far the majority of applications fall into this category.

Step #2 – Coding with the Workstation Application Generator Software

For many data collection programs, a software development tool called the Application Generator Software (Catalog No. 2708-NAG) may be all you need to implement your application. Application Generator Software is extremely useful in the development of applications, prototyping applications with an end-user, and creating demonstration programs quickly. Application Generator Software requires very little learning curve. It allows the program to be simulated on the PC, which helps in the testing and debugging of the program, and outputs a syntactically correct A-B BASIC source program as a ".BAS" file, which can then be compiled as described later.

Step #3 (Alternate) – Coding with QuickBASIC Editor

If you do not use Application Generator Software, it is recommended that you enter your A-B BASIC programs using the QuickBASIC editor. This provides syntax checking, subroutine and function menus, and liberal help screens. Since A-B BASIC is a subset of QuickBASIC, you can enter any A-B BASIC commands with the editor.

Note: When using the QuickBASIC programming environment, be sure to save your A-B BASIC program file in ASCII text format rather than the compact format. (Use the SAVE AS option in the QuickBASIC editor.)

Step #4 – Simulation Using Application Library

A design goal for A-B BASIC was to have the workstation behave like a PC when running the same code. Since the PC and workstation act the same, the ideal place to test the subroutines of your application is right on the PC itself in the QuickBASIC environment.

The A-B BASIC Development Procedure (cont'd)

In order to emulate programs on a PC the A-B Application Library was developed (see Chapter 5). Many basic I/O routines for the various devices found on a workstation are available through this library. These include inputs from barcode scanners, timeouts, formatted and edited keypad read routines, data conversion routine, etc. One of the clear benefits of using the library is that you don't have to "re-invent the wheel".

There are two main files in the Application Library: ENV.BAS and ENVPC.BAS. When testing your program on a PC, just include ENVPC.BAS into your program, and all your I/O will be directed at standard PC devices which will emulate the workstation. When it's time to run the program on the real workstation, simply substitute ENV.BAS at compile time.

Step #5 – Compiling the A-B BASIC Program

After the program has been written and tested on the PC using ENVPC or Application Generator Software, the program must be converted into a form that a workstation can understand. This is done with the A-B BASIC cross compiler (LXB). LXB will flag any lines that do not fit the subset of allowed commands. It will also flag syntactical errors, but those will normally already have been taken care of by the QuickBASIC editor. The output of the LXB compiler is a ".LXE" file, suitable for downloading to a workstation.

Step #6 – Downloading

You can download the program with another software product called Network Manager Software (Catalog No. 2708-NNM). Network Manager Software is a data collection and network management program which can collect data, download programs and files, and perform network management functions. It is the preferred tool for commmunicating between a PC and a workstation network. It interprets network status records and provides repetitive command transmissions to multiple workstations. However, in the absence of Network Manager Software, any asynchronous terminal emulator that has a file transfer feature and honors XON/XOFF protocol should work.

As soon as the download is complete, the program automatically starts running. At this point the program can be tested for proper functioning on the workstation itself.

To summarize, the steps in developing an application for a workstation are as follows:

1. Lay out a structured design for application
2. Decide if application is small or large (is CHAINing necessary?)
3. Use developed tools wherever possible (BASIC Language Development Kit)
4. Code the application using the QuickBASIC editor or Application Generator Software
5. Test the program under simulation (include ENVPC.BAS)
6. Compile with LXB (include ENV.BAS)
7. Download the program
8. Test the program on the target workstation

Loading and Auto-Starting an A-B BASIC Program

The .LXE "download" file produced the the A-B BASIC cross-compiler must be transmitted, through the Master Workstation, to the target workstation. This transmission must honor whatever protocol has been chosen for Host-Master communication in the Setup menus on the workstation: XON/XOFF, Polled Mode, Request Response, or whatever other method is enabled. In particular, the following **will NOT WORK**:

```
COPY PGM.LXE COM1
```

This WON'T WORK, since the standard DOS communication drivers do not implement XON/XOFF.

Downloading a BASIC program in .LXE format will terminate any program in progress at the start of the downloading process. Successful completion of a download results in the initiation of the new program. However, a program that is in .PGM format (described in Chapter 3) can be downloaded into RAM without interrupting the currently running program. The .PGM format program can later be executed by calling it with a CHAIN statement.

Start-up Condition of an A-B BASIC Program

This section summarizes the status of the workstation when a program starts.

LCD:

The front panel display may be thought of as being blank. The first character output will appear in the upper left corner of the display. The cursor is on.

The words "BASIC START" are placed on the display by the operating system before the BASIC program is initiated. If this text is visible long enough to be noticed, the probable cause is a BASIC program which has not yet reached a PRINT statement.

Random numbers:

The random number seed is set to 1.

RAM files:

All files are closed but their contents, if any, remain unchanged.

Variables:

Are 0 or null string per QuickBASIC defaults.

Host device:

The Host device is closed and therefore any data collected by the workstation and queued is held until an OPEN "Host" command is encountered.

Loading and Auto-Starting an A-B BASIC Program (cont'd)

Prefix digits 9 and 10 of the network record: are 01. (Refer to Attended Workstation User's Manual, Chapter 8 on responses to Network Directives.) Manual Intervention in an A-B BASIC program:

In a normal production use of a workstation, the application program remains running continuously. Consequently, only limited facilities are supported for manual intervention.

Sending an > > A command to the terminal will abort any A-B BASIC program in progress and generate the "operator abort" BASIC ERROR message. For more information on commands, refer to Attended Workstation User's Manual (Chapter 8).

The > > G command will restart a terminated A-B BASIC program. It is ignored if the program is already running.

Note: The program will restart at the beginning not at the point of termination.

Termination of an A-B BASIC Program

The following are reasons for the absence of a running A-B BASIC program:

1. The download of a .LXE file was never started, never completed, or resulted in an error. The load status display (number 7) should be consulted (refer to Attended Workstation User's Manual).
2. The program was started but terminated voluntarily by executing an EXIT, STOP, or SYSTEM statement, or by "reaching the bottom". We suggest that if you allow a program to terminate itself, you leave a descriptive message on the application status display (number 9) before termination. (See the STAT device later in this manual.)
3. A running program is automatically terminated when any record of a download (.LXE) file is received. This results in the BASIC ERROR "New Program Load", which means "Program aborted due to a new download".
4. The program was aborted by the operator using > > A.
5. Power failure.
6. A runtime error occurred.

A-B BASIC Run-Time Errors

All errors encountered by the A-B BASIC pseudo-code interpreter are serious enough to result in the termination of the application program. Program termination results in the following:

1. A return to the display of workstation status information (10 operator-selectable status displays discussed in the Attended Workstation User's Manual.
2. An error message is sent through the network to the Host which begins with the words "BASIC ERROR".
3. The error message is also available on the BASIC run-time status display (number 8), and through the read-only menu.
4. The diagnostic command < E will return the error message on demand (refer to Attended Workstation User's Manual, Chapter 8).
5. The error flag E is posted over the 8 on the summary status display (number 0).

Format of a BASIC Error Message

The format of a BASIC run-time error message is:

text offset date time

where **"text"** is an explanation of the error,

and **"offset"** is the displacement from the start of the program's pseudo code of the pseudo-instruction which was unable to complete. The corresponding BASIC statement can be found by correlating this value with the offset information provided in the program listing (.LST File) from LXB. The following chapter discusses how to get a listing file from LXB.

"date time" is the date and time the error occurred in the same form returned by the DATE$ and TIME$ functions. Example:

0101000600BASIC ERROR: New Program Load 054B: 9102281345

Runtime error messages presented to the Host computer are preceded with the customary 10 digit prefix, digits 9 and 10 of which are 00, followed by BASIC ERROR (refer to Attended Workstation User's Manual, Chapter 8).

Power Failure

Circuitry within a workstation is capable of detecting and reporting to the processor an imminent loss of power. When this occurs, all tasks are brought to a logical conclusion and the following message:

```
POWERFAIL SHUTDOWN!
```

is sent to the LCD. Normally, this message will never be seen because a true power failure will blank the display. Its viability for any perceptible length of time is evidence of a power failure which was not quite long enough to completely turn off the workstation, or a workstation overly sensitive to power fluctuations.

The restoration of power results in this environment for the A-B BASIC program:

1. The program is restarted FROM THE BEGINNING.
2. All files (RAM FILES) in existence at power failure are intact, although closed. Any write operation (a single PUT, etc.) to a file of a length under 4K bytes which was in progress when the power failure occurred, either did or did not complete AS A UNIT. Stated differently, record fragments are not present, only whole records.
3. All others program parameters are reset as for any other program initiation: variables 0, etc.

Programs wishing to distinguish between initial (cold) start-up and powerfail restart (warm), should open a file named, say, POWER. If the LOF function on POWER returns 0, assume a cold start and PUT something into the file. A non-zero file length implies a warm start.

Chapter 3

The A-B BASIC CROSS-COMPILER (LXB)

Overview

The QuickBASIC Development Environment alone will not generate a program for a workstation. QuickBASIC allows the developer to create a program which can execute on a PC, but it is unable to create programs that a 2708-DH5 workstation can understand. To do that, we have developed a cross compiler that accepts an ASCII text file containing valid BASIC statements and creates a file that will execute only on a 2708-DH5 workstation. The extension .LXE is an adaptation of the .EXE name commonly used for executables under MS-DOS.

Typically a program is created and tested on a PC compatible computer with QuickBASIC and the application library. Once the program is refined through editing and testing, it can be compiled into an LXE file. The LXE file is also an ASCII text file. However, because it contains pseudo-code it doesn't look like anything familiar. Even character strings will not be visible in the file. The LXE can therefore be considered an encrypted file.

Because it is pure ASCII displayable text, it can be uploaded to other computers without worrying about control codes or binary objects becoming garbled. There should be no problem, for example, uploading an LXE file to an IBM mainframe, which can later download the program to a 2708-DH5 network using its native EBCDIC/ASCII translation tables.

Each line of an LXE file begins with a two-character sequence called a Network Directive. These are discussed in detail in the Attended Workstation User's Manual. For downloadable files, the two characters are a greater than (>) and a comma (,). When the operating system sees an incoming record prefaced with these two characters, it knows that this is a line from a compiled program and not a line of data. If the file is in PGM format (see next section), it will have an additional network directive appended to the front of each record, and a 2-record header on the file.

The next two characters will begin with AA and increment sequentially through ZZ.

Any further discussion of the contents of the LXE records is beyond the scope of this document. Since the LXB compiler puts out an entire file, no further information is needed.

Following a successful download, the workstation will begin immediately executing the program.

Invoking the Compiler

The LXB compiler is executed on a PC by entering the following command. You must be in the right directory, or have the directory containing the LXB.EXE file set in your PATH. Optional parameters are in parentheses:

LXB param 1 (,param2) (,param3) (,param4)

The parameters are:

param1	Source filename (Default extension: .BAS)
param2	Download filename (Default extension: .LXE)
param3	Listing filename (Default extension: .LST)
param4	Secondary download filename (Automatic extension .PGM)

Entering the LXB command without any parameters will cause the compiler to prompt for the source, download, and list file names, offering defaults as appropriate. The default file extensions are in most cases the only allowable ones, and should not be specified or overwritten:

A.BAS source file extension is assumed. The source filename must always be specified.

The .LXE file is always generated. If the .LXE filename is omitted, it assumes the same name as the source file, and has the extension LXE.

If the .LST filename is omitted, the compiler will not generate a listing. Use a comma to hold its place if you want to specify a 4th parameter but don't want to enter the filename for the .LST file.

If the .PGM filename is omitted, the compiler will not generate a secondary download file. If it is specified, LXB will output (in addition to the regular .LXE file) a download file which contains the necessary network directives to cause the program to be stored in RAM and not immediately executed upon download. This feature is used for programs which will be CHAINed to later on. The RAM filename for the CHAIN statement will be the one specified in this parameter. In order for the fourth file to be generated, the .LXE file's default extension of ".LXE" *MUST NOT* be overridden.

Invoking the Compiler (cont'd)

Examples:

LXB

In the case above, the operator is prompted for 3 file names.

LXB Prog1

In this case, the file Prog1.BAS is read, and Prog1.LXE is the product. (This is the fastest compile option.)

LXB Prog1,Prog1,Prog1

File Prog1.BAS is the source, Prog1.LXE and Prog1.LST are generated.

LXB Prog1,Download,PRN

Generate Download.LXE, and send listing directly to the printer (PRN).

LXB Prog1,Startnow,Prog1,Wait

Generate Startnow.LXE and Wait.PGM output files, and Prog1.LST as the listing file.

A – Source File Specification

The A-B BASIC compiler requires a plain ASCII text file. By default, the QuickBASIC editor saves the source text in a compacted format. Be sure to use the SAVE AS option in Microsoft's QuickBASIC to save the source as an ASCII file. LXB cannot recognize the compacted representation.

For a tutorial on BASIC programming, please refer to Microsoft's "Learning and Using Microsoft QuickBASIC", and "Programming in BASIC", both of which are supplied with the Application Generator Software (Catalog No. 2708-NAG) and BASIC Language Development Kit Software (Catalog No. 2708-NBD). A list of the STATEMENTS & FUNCTIONS supported by A-B BASIC can be found in Appendix C of this manual.

Invoking the Compiler (cont'd)

B – Downloaded Executable File

The usual extension of a file that is produced by most other compilers is ".OBJ". Those files usually go through a separate LINK step before an executable ".EXE" file is generated. However, the extension of the A-B BASIC cross compiler output is ".LXE". The extension ".LXE" is used for two reasons:

1. The file cannot be link edited like a usual PC object file.
2. There is only one useful operation that can be performed on this file: Download it to a workstation.

The code portion of the file is checksummed. Except for carriage returns, the entire file is within the printable ASCII character set. Transmission of a download file using modems should be safe since a corrupted file will result in a checksum error instead of program initiation.

C – Listing File

If specified, LXB places a listing into a file with the same base name as the source file, with an ".LST" extension. The OFFSET column of the listing shows the location of the pseudo-code corresponding to the BASIC statement. Error messages at run-time provide this offset as an aid in locating errors.

Program pseudo-code size is given in the listing file which can be used to gauge how much of the maximum 13,700 bytes of available program space (32K memory workstation) were used. Other statistics provide insight into the free variable space and other elements of memory use.

D – Secondary Download File

If the secondary download file parameter is specified, then a .PGM file will be output in addition to the .LXE file. The .PGM file is a special type of download file which will not begin executing immediately, but rather stores itself into RAM for future execution via a CHAIN statement from another program. It does this by appending some additional network directives onto the front of the file, and onto each record.

Chapter 4

Special Devices in A-B BASIC

Introduction

Most I/O in A-B BASIC is performed by accessing special device names which are unique to the workstation. In A-B BASIC, as in QuickBASIC, devices are accessed with the same statements as files. For example, access to the workstation's barcode reader is obtained by an OPEN of the device named "BAR", followed by GETs or LINE INPUTs.

We highly recommend use of the BASIC Language Development Kit (Catalog No. 2708-NBD) of subroutines for faster, easier development of larger programs. Subroutines in that library automatically open the required devices and perform the appropriate timeouts and formatting operations. In addition, debugging can be done on the PC without the need for downloading each time a program change is made. Refer to Appendix C.

Most of the following information assumes that you are not using the BASIC Language Development Kit, but are writing a program which directly addresses I/O devices.

Here is a list of all reserved filenames which address the special devices. Note that they are different from DOS device names, and that they do not contain colons:

Real Devices	**Name**
Front panel LCD	LCD
Keypad	KEYS
Comm line - primary	COM
Comm line - auxiliary	AUX
Barcode	BAR
Lights on keypad	LITE
Timer	EGG
Host computer	HOST and NET

Pseudo Devices	**Name**
Queue to host computer	QUE
User status display	STAT

RAM files (any non-reserved name, 12 characters per file name max.)

Introduction (cont'd)

Some statements and intrinsic functions implicitly refer to a specific device. These are:

Front panel LCD	CLS, CSRLIN, LOCATE, POS, PRINT and WRITE (without file numbers)
Keypad	INKEY$, INPUT$ (without filenumber), LINE INPUT (without filenumber)
Beeper	BEEP, SOUND
Timer	DATE$, SLEEP, TIME$, TIMER
Files	KILL

Device: LCD Display

Reserved device name: LCD
These statements always access the LCD display:

PRINT and PRINT USING (without a file number)
WRITE (without a file number)
CLS
CSRLIN
LOCATE
POS
Prompts for LINE INPUT (without a file number)

As an alternative, OPEN "LCD" may be used to associate the device with a file number and PRINT # or WRITE # used.

Note: Once OPENed, the "LCD" file CANNOT be closed.

At program start, the LCD displays the words "BASIC START". The cursor is at the home position and is on. The LCD is set to clear when the program's first character is output to it. Thus, the programmer can act as if the LCD were actually already blank.

PRINTs to the LCD, ending without a comma or semicolon (, or ;) leave the cursor at the start of the next line. Thus, the next PRINT causes that line to be cleared before data is written.

PRINT statements terminated by a comma or semicolon leave the cursor where it was at statement's end, as shown in the following example.

Device: LCD Display (cont'd)

Note: See the description of PRINT USING and WRITE for details on other formatting options supported.

See also the READ.BAS collection of BASIC Language Development Kit subroutines for useful LCD output subroutines, including automatic formatting of display lines, automatic clearing of the display, and cursor positioning.

```
CLS
Name$ = "Fred"
PRINT "Please Enter Now!"
PRINT "Thank You"; Name$;   ' Keep cursor on second line.

CONST ScreenDev = 1
OPEN "LCD" FOR OUTPUT AS #ScreenDev   ' Cannot be CLOSEd
LineWidth% = LOF (ScreenDev) \ 2      ' Width of display.
PRINT #ScreenDev, "Please Enter Code:";   ' Cursor remains on line.
LINE INPUT, Code$
```

Device: Keypad

Reserved device name: KEYS
These statements always access the Keypad:

INPUT$ (without a file number)
LINE INPUT (without a file number)
INKEY$

When a Keypad line is being entered with LINE INPUT, only the enter and backspace keys perform the expected termination and correction functions.

As an alternative, INPUT$ and LINE INPUT can be used to access this device using OPEN "KEYS".

The BASIC Language Development Kit (Catalog No. 2708-NBD) modules READ.BAS and MENU.BAS contain several helpful subroutines for obtaining input through the Keypad.

```
CONST KeyDev = 1                        ' Assign device a file number
OPEN "KEYS" FOR INPUT AS #KeyDev        ' Open device.
DO WHILE a$ = " ":  a$ = INKEY$:  LOOP  ' Wait for a key stroke.
LINE INPUT "Enter your name:  "; Name$  ' Must press enter.
b$ – INPUT$ (1)                         ' Another way to get a key stroke.
```

Device: Barcode Scanners

Reserved device name:	BAR
Open Modes:	INPUT
Statements:	LINE INPUT
Functions:	EOF returns TRUE when a complete record has arrived. Once TRUE, a LINE INPUT statement can be used and will immediately return the data when LINE INPUT completes. EOF returns to FALSE. LOF remains 0 until a barcode is read. Then it contains the length of the data record. It must be called before LINE INPUT. Once LINE INPUT has been used, LOF returns 0 until the next barcode is read.

IOCTL$ may be called after LINE INPUT to obtain more information about the type and origin of the data. In general, the string which is returned is of the form:

Type/Source

When reading barcodes:

Type can be: C39, I25, UPC, CBAR, or C128
Source can be WAND or LASER

There are two intrinsic functions which may be used before LINE INPUT to determine whether a barcode has been read: EOF, and LOF.

For barcodes, “read” means that the operator correctly used a wand, laser, or slot reader to scan one of the barcode types enabled through the barcode menu. For example, an A-B BASIC program will not be aware that a UPC barcode has been scanned if UPC is not enabled or if another condition such as a check digit has not been satisfied. It also means that the operating system was able to decode the pattern of the barcode because the reader was operated at the correct speed and angle. Unsuccessful attempts to read a barcode are not reported to the BASIC program.

Once a read has occurred, the TRUE returned by EOF, the barcode length from LOF, and the data from LINE INPUT, are “latched” until a LINE INPUT is used.

Note: Allen-Bradley slot scanners (Catalog No. 2755-B1, -B2) can be used in place of a wand or hand-held scanner.

Device: Barcode Scanners (cont'd)

This means that:

1. EOF and LOF may be called any number of times before LINE INPUT, and they will return the information for the current barcode. Program structure might make it convenient for one procedure to detect the presence of a barcode with EOF, and another to get its length with LOF, and yet another to place its contents into a string with LINE INPUT.
2. All barcodes must be accepted by LINE INPUT before subsequent barcodes can be read! It is not possible to use LOF to read a barcode, then to ignore it because its length is wrong.
3. Once LINE INPUT is used, EOF and LOF are reset to false and 0 until another barcode arrives.

The status returned by IOCTL$ is valid from one transition from EOF TRUE to another.

To summarize, if you do not use the BASIC Language Development Kit subroutines (which are a faster and easier way to develop larger programs), the proper programming sequence to read barcodes is:

```
Optional test for EOF and/or LOF
Optional IOCTL$
LINE INPUT
Optional IOCTL$
CONST BarDev = 2
DO WHILE NOT EOF(BarDev): LOOP   'Wait for data.
SOUND 1200, 1 'Give a nice beep.
LINE INPUT #BarDev, InputData$
DataLength = LEN(InputData$)
```

Device: Host Computer

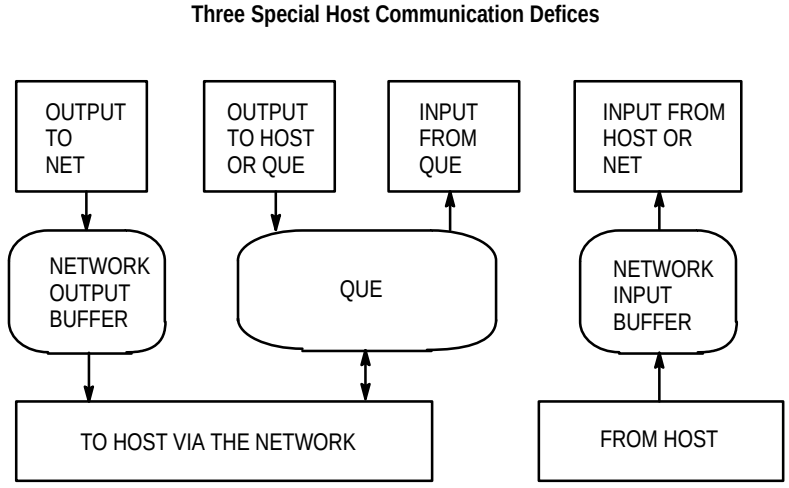
Reserved device names: HOST, NET, and QUE

It is very easy to forward the data collected by a workstation to the host computer attached to the Master Workstation. We recommend the BASIC Language Development Kit subroutines SendQue and Send. However, if you are not using these, the simplest method is to OPEN the “file” named HOST and WRITE # or PRINT # to it.

Two important features are that:

1. The data is forwarded to the host computer regardless of the network role played by the unit. Stated differently, output to HOST works the same for a workstation which is a Master, Concentrator, or any other workstation type.
2. Once a record has been written to HOST, no further action by the A-B BASIC program is required to get the information to the host. If the workstation is offline, the data is stored temporarily in a FIFO queue and forwarded when network conditions permit. This occurs transparently to the application.

Figure 4.1
Data I/O through a Network



This diagram shows the various methods which can be used to move data into and out of a workstation through its network.

All output operations to HOST enqueue data in a FIFO (first-in-first-out) queue, which is a special form of RAM file. The network task removes records one at a time as it is able to transport them to the host computer.

Output operations to the NET device bypass this queue. The record is placed in an output buffer which is checked before the queue. Use this method sparingly for high priority alarms which must bypass queued records.

Device: Host Computer (cont'd)

Note: There is no queue for data FROM the host, only a single buffer. Input operations from HOST and NET access this buffer.

As just mentioned, the FIFO queue is a special file. As such, it can be manipulated as a file, through the reserved file name QUE.

IMPORTANT NOTE: Outputs to the QUE device will remain in the queue and will not be sent to the host until the QUE device is closed and the HOST device is opened. Nothing further needs to be done with the HOST device; it only needs to be left open until the entire queue is emptied.

If, on the other hand, data should be routed to the host as quickly as possible, do not use the QUE device at all, but rather send everything directly to the HOST device. It will still use the queue, but will not wait to transmit.

General restrictions and warnings:

1. Only printable ASCII characters may be moved through the network. Output only characters within this range:

   Lowest valid: Space, CHR$(32), Hex 20
   Highest valid: }(right curly bracket), CHR$(125), Hex 7D

   Programs which access QUE as a binary file will also encounter carriage return/line feed sequences. These have been inserted by the interpreter. Do not remove them or try to insert them explicitly using CHR$( ).
2. Like any other RAM file, the size of QUE and HOST is limited by physical RAM. Use the FRE function often to prevent out-of-memory conditions which could have disastrous effects on operations.
3. All output operations must contain a complete record in ONE statement. All outputs place a trailing carriage return/line feed sequence after the data provided. It is not possible, for example, to use multiple PRINT statements, terminated by a comma to assembly a HOST or QUE record piecemeal. THIS RESTRICTION IS ENFORCED, see PRINT.
4. Do not allow HOST and QUE to be OPEN at the same time.
5. Input FROM the host computer may not begin with < or > or a numeric digit (0 to 9). The Attended Workstation User's Manual, Chapter 8 describes the effect which these characters have on the firmware.
6. All output to NET has the potential disadvantage that its data might be classified as a network status or other error message and not application related data. This can happen because, when using the NET device, the "Transaction Code" digits 9 and 10 of the 10 character record header are set to zeroes, and cannot be changed by an IOCTL statement (see IOCTL description later in this chapter). Two zeroes are typically an indication of a status or diagnostic message, BASIC ERROR, or other system generated response.

Device: Host Computer (cont'd)

For example:

```
PRINT #NETDEV, "BAD INVENTORY NUMBER"
```

may be trapped by the host computer as a BASIC ERROR because the complete record sent to the host will look something like this:

0101000100BAD INVENTORY NUMBER

If the host program is testing for zeroes in digits 9 and 10, and is then looking for "BA" to indicate a BASIC ERROR, it will become confused by this error message, and potentially abort the programs.

We suggest that, if you must use NET instead of HOST, make your first two bytes something odd like **.

7. A maximum length is imposed on all records moving through the network. Consult Appendix D in this manual.
8. AnyA-B BASIC program reading from HOST or NET must do so at a rate fast enough to "Keep up" with the host. If a new record arrives before the previous one is input, the "lost" count is incremented on the #5 status display.

Details of Specific Statements and Functions

We highly recommend that you utilize the subroutines contained in the BASIC Language Development Kit (Catalog No. 2708-NBD) to do all I/O in the workstation. These library routines allow you to simulate the workstation on a PC during development, saving you hours of time and headaches. When using the BASIC Language Development Kit, you will only need to call OpenLINX, then execute a ReadEvent and wait for a return. Any data received from the devices (slot reader, COM ports, etc.) will be returned in appropriate variables, and flags will be set to let you know where the data came from. Refer to Appendix C for details. If you must use the devices directly, the following programming examples will show you how:

OPEN "HOST"

Host should be opened only as a sequential or random access file, in keeping with the record oriented nature of network I/O. QUE may be OPENed for binary access so long as the general restrictions are observed.

An OPEN "HOST" has a very important side effect: it allows the network task to deque records from the QUE and forward them to the host.

Program initiation CLOSEs HOST. Program termination leaves HOST unchanged.

Details of Specific Statements and Functions (cont'd)

CLOSE "HOST"

Note: When HOST is CLOSEd, the A-B BASIC interpreter always tells the network task that no data is available from the QUE, even if data is really present. There may be records remaining in the QUE. If so, they will not be available to the network until HOST is once again made OPEN. If necessary, use the LOF function to determine when the QUE is empty, before CLOSEing HOST.

WRITE

If the presence of commas and quotation marks is acceptable to your host, this is the simplest way to send a record. Don't forget to precede the WRITE with a call to FRE to make sure memory is available. As with all output to the network, the interpreter adds the required record delimiters.

PUT to HOST

PUT is an excellent way to output a variable which has been defined in a TYPE statement. Just remember that all the elements in the TYPE must be fixed length strings. Numerics are not allowed (as binary quantities) since adherence to the "printable character" restriction is not assured.

The data is put into the queue for automatic transmission to the host computer, in order, when possible.

PUT to NET

If the (network) output buffer is empty (see LOF), the record provided will be the next one taken by the network for transmission to the host.

Note: If the output buffer is in use, execution of this statement suspends the A-B BASIC program until the previous record is taken. So, unless your program is prepared to wait, do not use this statement without first checking LOF on the NET device. Better yet, PUT to HOST.

PUT to QUE

This statement acts like PUT to HOST. Generally, it is used to create a "batch" of records for the host which will be released at some future time by an OPEN "HOST". Important distinction: PRINT, and WRITE to the QUE add a record delimiter in the same way as output to HOST. Outputs to regular RAM files do not perform this insertion.

PRINT

All PRINT #s to HOST, NET, and QUE are restricted. After every PRINT, the interpreter inserts the carriage return, line feed record delimiter. It is not possible to use multiple PRINTs to make one record.

Details of Specific Statements and Functions (cont'd)

LINE INPUT # for HOST and NET
GET # for HOST and NET (OPEN with RANDOM)

These statements will read one record which the host computer has sent to this. If there is no record available, the program waits for its arrival. Use the EOF or LOC functions first if waiting is not acceptable.

LINE INPUT # for QUE
GET # for QUE (OPEN with RANDOM enforced)

These statements act as if a file is being accessed because QUE is a file.

EOF for HOST and NET

EOF returns TRUE when a record is available for reading. When FALSE, nothing is present.

LOC for HOST and NET

LOC returns the number of characters which can be read from the network input buffer.

Example:

```
IF LOC(#1)  0 THEN
      LINE INPUT #1,  INLINE$
ELSE
      ' Do something else
END IF
```

LOF for HOST

LOF returns the size of the queue, that is, the number of bytes awaiting transmission to the host.

LOF for NET

LOF returns the number of bytes free in the network output buffer. When non-zero, a single record of that size may be output without waiting. When 0, output is permitted but the A-B BASIC program will wait until the previous record is accepted by the network. Note that only one record may be output even though the record is shorter than the buffer. For example, say the function returned 200. It is not possible to output 3 records of 20 bytes without waiting.

Details of Specific Statements and Functions (cont'd)

IOCTL$ (intrinsic function) for HOST and NET

This function returns a string formatted as follows:

Byte 0: '0' when the workstation is offline to the network.
'1' when the workstation is online.

Bytes 1-4: is the workstation's number (without a minus).

Bytes 5-9: is the workstation type (its network role) as

NORML	CONCN
MASTR	SUBM
ALTM	ALTSM

IOCTL (statement) for HOST

Characters 9 and 10 of all records arriving at the host contain a 2 digit number ("Transaction Code"). Numbers 00, 98, and 99 are reserved. A-B BASIC programs can change these two digits for records from HOST, from the default of 01.

For example, if the statement IOCTL #5, "29" is used where #5 is associated with HOST, digits 9 and 10 of records containing data from this workstation, will be 29.

Note: Only numeric digits are valid. IOCTL is not valid in its statement form for the NET device.

Devices: Communication Ports, Primary and Auxiliary

Reserved device names: COM and AUX
Opening a com port:

Access to the communications ports begins with the statement OPEN "COM" and OPEN "AUX". On a PC, QuickBASIC allows you to specify baud rate and other parameters in the OPEN "COM" statement. In A-B BASIC these are NOT valid! Comm parameters are set using the Setup Menus (see Chapter 5 for details). If you are using the ENV/ENVPC collections of subroutines in the BASIC Language Development Kit (Catalog No. 2708-NBD), a call to OpenLINX will automatically open all required comm ports.

The primary communication port is dedicated to networking functions in all terminal types except NORMAL. Any attempt to OPEN 'COM" on any workstation which has not been set to a workstation type of NORMAL (using the Setup Menus) will result in a runtime error.

Units with a second ("AUX") port may use that port via OPEN "AUX" regardless of the workstation's network role.

OPENing a communications port has no effect on the state of the modem control lines DTR and RTS. These are always under the control of the BASIC program using IOCTL, or one of the BASIC Language Development Kit subroutines discussed later.

Devices: Communication Ports, Primary and Auxiliary (cont'd)

A – Output to a Communication Port

The BASIC Language Development Kit (Catalog No. 2708-NBD) contains subroutines SendCom, ReadEvent and Test Event for accessing the COM port when the terminal is configured as a NORMAL workstation type. If you need more immediate control of the port, use the following programming instructions:

The PRINT and WRITE statements are available for output to the communications ports. Unless a semicolon is placed at the end of the PRINT statement, a carriage return will be output. When enabled through the menu, a line feed is also output.

Each of the communications lines has an output buffer. The LOF function returns the number of bytes free in this buffer. When less than the size of the message, output to the corresponding port will result in a wait until all characters can be enqueued. This wait will be short in a workstation which is able to transmit. However, for a workstation which has received an XOFF, the wait can be long (or even infinite)!

B – Input from a Communication Port

All COM and AUX input is performed with the INPUT$ function. If more characters are requested than have been accepted by INPUT$, the program waits until the count is satisfied.

LINE INPUT is not valid with the COM or AUX device.

The LOC function returns the number of bytes currently in the port's communication input buffer bytes. A program will never wait if LOC is used to get the byte count, and that count is used in INPUT$.

For example:

```
CONST AuxDev = 1
OPEN "AUX" FOR INPUT AS #AuxDev    ' Cound also be "COM"
J$ – INPUT$ (LOC(5), #5
```

NOTE
Receipt of Control-C (or whatever control code was specified in the Setup Menu) will cause the workstation to enter the menu mode. Receipt of Control-R will reboot the workstation.

Devices: Communication Ports, Primary and Auxiliary (cont'd)

C – Modem Control Lines

The primary communication port is equipped with DTR and RTS. These can be set through the IOCTL statements. The character string passed must contain two bytes. There are two valid options for each byte:

"0" to turn the line off
"1" to turn the line on

The first byte controls DTR, the second controls RTS. Thus:

```
IOCTL #5, "01"
```
turns off DTR and turns on RTS.

These modem lines can also be controlled using the BASIC Language Development Kit routines SetDTR and SetRTS.

AUX has only one line, DTR. It is controlled in the same way as COM. For future compatibility, always add a second (character) zero, to imply RTS off.

Device: RAM Files

Name: Any name which is not reserved and is within the rules for file names.

Description:

Data files on a PC are resident on disk. They are RAM resident in the workstation. A-B BASIC supports the three types of file access provided with QuickBASIC, sequential, binary, and random. File I/O is thoroughly discussed in Microsoft's "Programming in BASIC", Chapter 3.

Statements and Functions supported:[1]

CLOSE	LOF
EOF	OPEN
GET	PRINT
INPUT$	PUT
KILL	SEEK Statement
LINE INPUT	SEEK function
LOC	WRITE

```
CONST DataFile = 1
OPEN "EMPLOYEE" FOR OUTPUT AS #DataFile
PRINT #DataFile, "Employee  ";EmpName$ ;" – ";TIME$
CLOSE #DateFile
```

WARNING:

Both QuickBASIC and A-B BASIC allow a PUT to a record number which is greater than the last record in existence before the PUT. Both BASICs extend the file to the length required to accommodate the request.

- Refer to Appendix M for any restrictions on Syntax.

Device: RAM Files (cont'd)

For example, in a new file:

```
PUT #X, 1, something
PUT #X, 1000, something
```

will result in space being allocated for 1000 records, not 2!

Moral: *Keep record numbers under control.*

File Memory Management

Files are allocated in blocks of RAM. Every file has at least one free byte. In essence this means that a file with a length of zero is allocated one block. Each block contains 2 bytes of overhead. The remainder contains the file's contents. The block size is set via the configuration menu, to 256, 512, 1K, 2K, or 4K bytes. In an application with several small files, a block size of 256 is recommended. In an application with one large file, a block size of 4K allows for a slightly larger file and slightly better performance. The maximum file size is limited by a workstation's memory, not A-B BASIC.

QuickBASIC does not allow you to PUT a variable length string to a RANDOM file. In A-B BASIC this is also not recommended. However, it can be done. Remember to allow at least 2 bytes of overhead for the carriage return/line feed sequence that accompanies each variable PUT to a record. Integrity of data stored and retrieved in this fashion cannot be maintained or updated with any relative degree of confidence.

Device: Beeper

This device is accessed through the BEEP and the SOUND statements. SOUND requires both a frequency and a tone duration. The correspondence between the frequency and a "true" pitch of that value will be approximate.

The duration of the tone is given in units of 1/20 second.

Unlike QuickBASIC which pauses until a SOUND is finished, A-B BASIC initiates a SOUND and continues. A second SOUND or BEEP which is encountered before the first completes, resultls in the truncation of the initial tone.

Device: Front Panel LED's

Device: Front Panel LED's
Reserved device name: LITE

The LEDs on the keyboard are numbered from 1 up. The maximum LED number depends on the keyboard style. PUTting a "0" to record number X, turns off LED number X. PUTting a "1" to record number X, turns it on.

Device: Front Panel LED's (cont'd)

The LED in the lower left corner of alphanumeric keyboards is used as a shift lock indicator by the operating system. A-B BASIC programs should not change it.

The BASIC Language Development Kit contains a subroutine SetLED which can be called to turn LEDs on and off. Or use this code example:

```
CONST LiteDev = 1
On$ = "1": off$ = "0"                ' Cannot be a CONSTant
OPEN "LITE" FOR RANDOM AS #LiteDev
FOR X = 1 TO 10                      ' Do 10 LEDs
    PUT #LiteDev, X, On$             ' Turn LED on.
    SLEEP .25
    PUT #LiteDev, X, Off$            ' Turn LED off
    SLEEP .25
NEXT X
CLOSE LiteDev
```

Device: The #9 User Status Display

Reserved device name: STAT

Description:

The front panel status displays shows information about the condition of a workstation. The number 9 status display is dedicated to showing status information from the application.

Two lines are available with 40 characters each. A summary status character which appears over the 9 on the zero summary status display, is also available.

The PUT statement is used to place information in the buffer which is shown when the #9 display is selected.

PUT to record number 1 places the string on the top line of the display. Put to record number 2 places the string on the second line of the display. PUT to record 3 places the first character of the string over the 9 on the summary display.

```
OPEN "STAT" FOR RANDOM AS #3
LTOP$ = "Data Corp Inventory"
LBOTTOMS$ = "Duncan Street Warehouse"
SC$ – " . "
PUT #3, 1, LTOP$
PUT #3, 2, LBOTTOM$
PUT #3, 3, SC$
```

If your application must shut down or cannot start up, for example due to a missing file that the host was supposed to download, put an error on the status display before program termination.

Remember that these statements do not cause the status display to appear, they only define the display's contents when invoked by the operator.

Device: The Egg Timer

Reserved device name: EGG

Description:

The egg timer is a word of memory which, when non-zero, is decremented by one every 1/100 second.

These statements, when used with the file number associated with an OPEN "EGG", control this timing device:

SEEK statement: sets the timer to an integer or long

For example:

```
OPEN "EGG" FOR INPUT AS #4
SEEK #4,  1000
```

sets the timer to reach 0 in ten seconds.

SEEK function: returns the current value of the timer as a LONG value. (LONG is used to avoid sign confusion for values over 32767 with an INTEGER).

LOF and LOC also return the timer's current value.

EOF returns TRUE (-1) when the timer "goes off" by reaching zero. This device is an easy way to implement delays and timeouts without the midnight rollover problem associated with TIME$ and TIMER.

Chapter 5

A-B BASIC Application Library

Introduction

The BASIC Language Development Kit (Catalog No. 2708-NBD) consists of a set of BASIC source files which you may use to decrease your development time and enhance the quality of your A-B BASIC applications.

Note: Subroutines for the A-B BASIC Application Library can be found in Appendix C of this manual.

The core of the library is the module ENV.BAS.

ENV.BAS provides a set of I/O routines for access to the workstation devices such as the bar code port, comm ports, network, and so on. Since virtually all workstation applications are driven by external events from multiple devices, ENV provides a simple mechanism for reading from more than one I/O device at once with timeouts.

In order to speed developoment on IBM PC compatible computers, we have provided a second library of subroutines (ENVPC.BAS) with names identical to those in ENV.BAS, but which use standard PC devices to emulate 2708-DH5XX workstations. This allows testing of A-B BASIC applications under MicroSoft's QuickBASIC. ENVPC allows you to simulate input from slot readers, comm ports, and the network all on your development PC without needing an actual workstation.

READ.BAS provides a library of field oriented I/O subroutines. You may read string fields, autoexit fields, numeric fields, secured (password) fields, and fields with default values using the routines. The FORMEX.BAS example program demonstrates forms style entry using READ.BAS.

MENU.BAS subroutines allow you to create visually oriented selection menus for easy-to-use application programs. This is also demonstrated by the example program FORMEX.BAS.

The IOTEST directory contains examples for A-B BASIC Low Level I/O. Scientific math support for process control applications is provided with MATH.BAS, TRIG.BAS, and HYP.BAS.

Using the Library

Installation consists of installing the diskette in drive A: and typing:

```
A:
INSTALIB d:
```

Where d is the destination drive.

You must first set up QuickBASIC, see Microsoft manual “Learning to Use Microsoft QuickBASIC” for installation procedures (manual is provided with the software). Also, make sure that the commands QB and LXBC are available from the installed directory “ \ LIB”.

The A-B BASIC compiler requires that all of your program reside in a single file. (It does not support a LINK step which would combine multiple independent object files into a single run file.) Since the BASIC Language Development Kit is made up of multiple files, the normal way to use it is to append all of the desired modules together and then load the resultant file under QuickBASIC or compile it with the A-B BASIC Compiler.

The only required module of the library when testing on the PC is ENVPC.BAS.

Suppose you have the following program:

```
DEFINT A–Z   ' Recommend default variable type
OpenLINX     ' REQUIRED library initialization
TIMEOUT% = 1000                    ' Set a 10 second timeout
DO
    CLD
    PRINT “Library Test”
    PRINT “Do something” ;         ' 2nd line MUST have;  or scrolls
    e = ReadEvent% (2)             ' The mask (the 2) means read
    SELECT CASE e                  ' from keyboard/bdg with timeout.
        CASE TimeoutEvent          ' The 10 seconds expired so make
            CLS: PRINT “ *Timeout* ”  ' this obvious.
            BEEP : SLEEP 2
        CASE BadgeEvent : CLS      ' The event that occurred was a
            PRINT “Badge read”     ' Bar code badge.
            PRINT InBuf$;          ' The badge contents is in InBuf$
            BEEP : SLEEP 2         ' Make some noise, and show badge.
        CASE ELSE : CLS            ' Case Else required in LinxBASIC!!
            PRINT “Key = ” ;e;     ' Display ASCII code for the key
            SLEEP 2                ' or negative code if its special.
    END SELECT
LOOP                               ' Do this test forever
```

Using the Library (cont'd)

If you don't want to type this in, you will find this sample program under ENVEX1.BAS. Now, assuming that the QB command (QuickBASIC) is in your directory or is set up in your DOS PATH for you to access it, you could enter the following DOS commands:

```
TYPE ENVPC.BAS > MAIN.BAS
TYPE ENVEX1.BAS > > MAIN.BAS
QB/RUN MAIN.BAS
```

The first TYPE command copies the ENVPC library source file into MAIN.BAS. The second type command appends your test program to MAIN.BAS. Finally, the command QB/RUN MAIN.BAS tells QuickBASIC to run and start executing MAIN.BAS.

A batch file is provided which you can use to do the DOS commands which were shown above. This batch file is supplied on the distribution diskette. To use it, type:

LQB ENVEX1

You will note that LQB.BAT also includes READ.BAS and MENU.BAS into the MAIN.BAS which it creates. If you do not need these modules, remove the TYPE lines for them from the batch file.

When you type the QB command, the QuickBASIC screen should be displayed and the bottom line will display "Loading and parsing" for a few seconds. Then it will say "Binding", and then the I/O environment emulator will be running, with your program executing.

Type ALT X or TAB then X to stop the Emulation.

Writing Programs

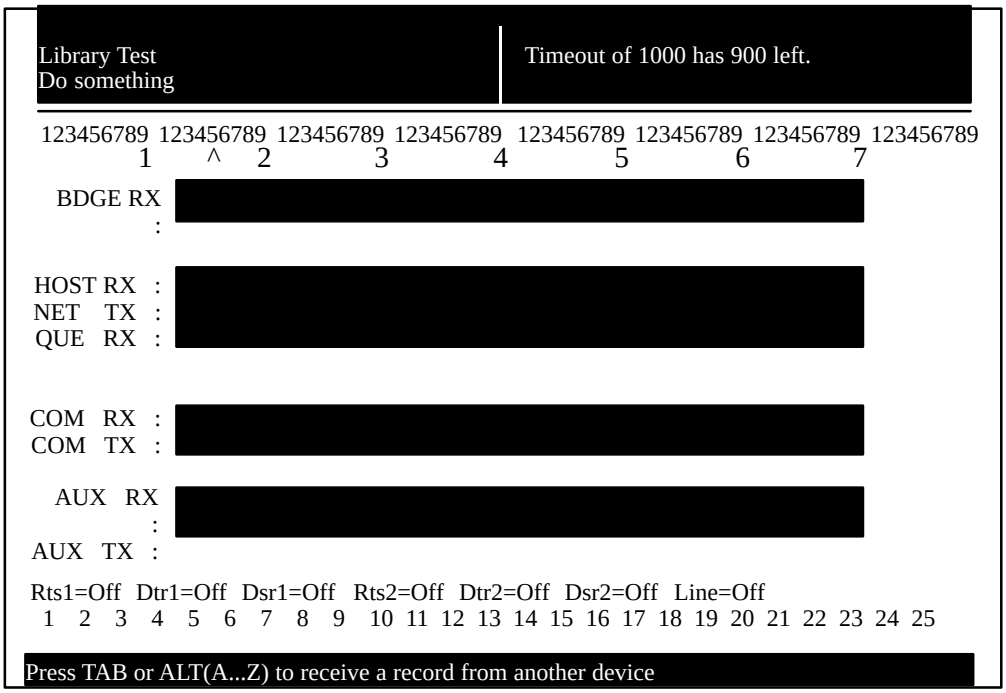
After examining the example BASIC programs and running them under the emulator, you should be ready to write your own. The subroutines are listed in Appendix C, along with the required arguments and returned values. Start with a small application and refer to the source code of the following files: FORMEX.BAS, ENV.BAS, READ.BAS and MENU.BAS for info about how to use each routine. Also print copies; of any BASIC files which interest you, as these will provide insight into various ways to use A-B BASIC.

When you are ready to test your program on the workstation, you should first compile it using the A-B BASIC Cross-compiler LXB. The LLXB.BAT batch file has the same syntax as LQB, except it starts with ENV.BAS instead of ENVPC.BAS.

When having subtle syntax problems, try examining MAIN.BAS or using LXBC to create a listing of MAIN.BAS.

Using the ENVPC Simulator

Figure 5.1
ENVPC Main Screen (Running ENVEX1)



TAB X or ALT X:	TERMINATES emulation.
CTRL-BREAK:	Pauses program and enters QuickBASIC environment.
TAB S or ALT S:	Slot reader manual input. You key in a record which will be read as slot reader input.
TAB B or ALT B:	Same as above (B is for Badge reader input).
TAB H or ALT H:	Host manual input. You key in a record which simulates a record from the host computer. Leave the station number off of the record.

If your actual host would send:

“1-23:REJECT/Bad lot number”

To workstation 23 attached to Master 1, you should just enter:

“REJECT/Bad lot number”

Using the ENVPC Simulator (cont'd)

Since that is all that your BASIC program would input.

TAB C or ALT C:	Com port manual input. You key in a record which simulates an input record from your RS-232 or RS-422 com port. Note that Master, Submaster, and Concentrator workstations cannot use the com port since it is in use by the network.
TAB A or ALT A:	Aux port manual input. A manual input record from RS-232 or AUX port.
TAB D or ALT D:	Puts the display in 40 column mode (FOR COLOR MONITORS ONLY!). The next TAB or ALT command will switch back to 80 column mode.
TAB L or ALT L:	Starts or stops logging. When logging is enabled, all output to all devices (except the display and front panel LEDs), and all input (except from the keyboard) is recorded in a file (usually named LINXIO.LOG). The log file contains a timestamp and device name for each I/O record.
TAB F or ALT F:	Opens a device input file. The Barcode, Host, Com, and Aux devices may each have an open device input file. Each time you press ALT N, the next record can be input from that file instead of keying in the records by hand. This can be a big time saver when simulating a complex application. You will be asked for the device that you want to input for (BC, H, C, or A) and you select the letter. You are then asked for a file name to input from. Make sure you enter the correct name or the emulator will terminate.
TAB N or ALT N:	If only a single device input file is open, then this will perform a device read from that file. If more than one device input file is open, then you will be prompted for the device which you want to input for (BC, H, C, or A).

A – Predefined Constants

Predefined constants are available in Env.BAS and EnvPC.BAS.

TRUE and FALSE may be assigned to integer variables which are used as boolean variables.

Using the ENVPC Simulator (cont'd)

The lower ASCII characters from 0 to 31 and 127 are named. These are:

NUL = 0	BS = 8	DLE = 16	CAN = 24
SOH = 1	HT = 9	DC1 = 17	EM = 25
STX = 2	LF = 10	XON = 18	SB = 26
ETX = 3	VT = 11	XON = 19	ESC = 27
EOT = 4	FF = 12	XOFF = 20	FS = 28
ENQ = 5	CR = 13	NAK = 21	GS = 39
ACK = 6	SO = 14	SYN = 22	RS = 30
BEL = 7	SI = 15	ETB = 23	VS = 31
DEL = 27			

The function keys are named F1..F10. The SHIFTED and ALT function keys are available as ShiftF1..ShiftF10 and AltF1..AltF10.

Alt letters are available as AltA..AltZ. Note that ENVPC uses the Alt keys for control.

Special key equates are:

ClearKey, ExitKey, EnterKey, Up, Left, InKey, OutKey, Right, and Down

Note: Left is identical to BS (backspace) on the workstation.

B – Global Variables

InBuf$: . . . . . Filled when SlotEvent WandEvent occurs
InCom$: . . . . . Filled when ComEvent or AuxEvent occurs
InNet$: . . . . . Filled when NetEvent occurs
DSPWIDTH%: . . . . . Width of the display in characters (40/16) (set in OpenLINX)
ONLINE%: . . . . . Set to TRUE when Master is polling (set by TestEvent%)
MEMFULL%: . . . . . Set to TRUE when File or QUE memory is almost gone
PENDING%: . . . . . Set to TRUE when a Send will pause
COMOPEN%: . . . . . TRUE if this is a normal workstation (set in OpenLINX)
TERMNUM%: . . . . . Number of this workstation (set in OpenLINX)
TERMTYPE$: . . . . . Type of workstation (set in OpenLINX)
EXITEVENT%: . . . . . Event code of the last event from a Read%
EXITOFFSET%: . . . . Offset in a menu or string from ReadStrField% and Menu%

Using the ENVPC Simulator (cont'd)

MODIFIED%: . . . . . True when a read changes the default value
TIMEOUT%: . . . . . YOU set to 100th seconds before a timeout for reads
BLOBALMASK%: . . YOU set mask used for ReadStr%, ReadStrField%, ReadInt% etc.
GLOBALATTR%: . . . YOU set attributes use for ReadStr%, ReadInt%, etc.
INSERTMODE%: . . . YOU set FALSE to disable insert mode on Read...

C – BASIC Language Development Kit Limitations

The BASIC Language Development Kit (Catalog No. 2708-NBD) subroutines provide access to all of the devices supported by the workstation. The following are some of the I/O methods that they do not support.

If ENV is unable to provide exactly the kind of support you require, MODIFY BOTH ENV.BAS AND ENVPC.BAS to add the special support you need. This way you can continue to debug your program under the QuickBASIC Environment before you download it to the workstation.

You *don't have* to use ENV or any part of it. We recommend it because it greatly speeds development and provides a common base for many workstation applications.

D – PC Simulation Constants and Variables

Note: Device numbers #20 and up are reserved for use by ENV.BAS.

PcMode . . . . . is TRUE when running under the emulator (ENVPC.BAS)
DSPWIDTH% . . . . . is set to 40

EmAltAccess is TRUE. Set it to FALSE to allow your program to access ALT keys. These are normally trapped by the emulator for device I/O access.

Bar code data is collected with the subroutine TestEvent%. The subroutine will return a BadgeEvent when the data has been read into InBuf$. Examine BadgeType for barcode inputs to get the bar code type ("C39", "I25", "UPC", "CBAR", or "C128").

Using the ENVPC Simulator (cont'd)

Display and Keyboard

You may use the full range of PRINT, LOCATE, etc. functions to write to the display. Do not use the cursor size arguments to LOCATE, but you can turn the cursor on and off using the third argument.

Reading from the keyboard should only be done using TestEvent% or one of the functions which use it such as ReadEvent%, ReadStr%, ReadInt%, Menu%, etc. The keys are returned as the ASCII value of the key. Special keys such as function keys and cursor keys, will return a negative event code. All special keys have named constants in ENV.BAS which MUST be used to test for a key value.

Network I/O

Only LINE INPUT and LINE OUTPUT are supported via TestEvent% as a NetEvent. The data is left in InNet$. Output to NET is supported with the Send subroutine. (Note that, as with any output to NET, you must first test to make sure the previous output to NET has completed, or you will get an error. This means the host must be online. If you need to send data while the workstation might be offline from the host, use SendQue instead.) Output to the network through the queue is supported via SendQue. No support is provided for opening the QUE device and accessing it as a file. No support is provided for PRINT # USING to the network.

COM and AUX

Only LINE INPUT is supported for input via TestEvent%. The events are ComEvent and AuxEvent, and the data is left in InCom$.

For OUTPUT you build a string which is output in exactly the form you provided it (Line-Feeds are added to Carriage Returns when that option is enabled). No support is provided for PRINT # USING to the COM or AUX ports.

Support for the control lines is provided with SetDTR and SetRTS.

Support for the status line is via GetDSR%. No event support is provided for DSR state changes.

Note: While the BASIC Language Development Kit subroutines will accept the AUX port as a target for DTR, RTS, and DSR commands, the WORKSTATION DOES NOT HAVE THESE LINES on the AUX port, and the results are unpredictable!

Using the ENVPC Simulator (cont'd)

Status Display

The application status display (one of the special status displays described in the Attended Workstation User's Manual) is not accessible through any subroutines in ENV, and is not emulated in ENVPC. Use a section of code which is conditional on the value of PcMode to set the status display.

Printing Reports and Forms

Since PRINT # USING.. is not supported, we recommend that you format reports by first printing the report to a scratch file, then copying the file to the device. Example:

```
listfile$ = "SCRATCH"
OPEN listfile$ FOR OUTPUT AS #1
PRINT #1, USING " . . . . . .
. . .
build your report, form, or complex response here
. . .
PRINT #1, USING " . . . . . .
CLOSE #1
OPEN listfile$ FOR INPUT AS #1
DO WHILE NOT EOF(1)
LINE INPUT #1,  s$
SendQue  s$          ' This could be SendQue, Send, or SendCom
LOOP
CLOSE #1
KILL listfile$
```

Appendix A

A-B BASIC and QuickBASIC Tips

Misspellings

Don't misspell names. BASIC is unforgiving of such errors since it will just create a new variable which is zero or blank.

Strings in TYPEs

Strings which are defined in TYPE statements are of a fixed length. These are always blank padded on the end.

Variables Beginning with "FN"

Do not start any variable name with FN, Fn, Fn, or fn. BASIC treats these as a reference to the archaic DEF FN.... mechanism and produces a syntax error.

Accidental Omission of %, &, ! # OR $

If you leave the trailing character off of variable names, you will usually get no warning from BASIC. Instead, the variable will be a different one than what you expected. Watch out for this! If you happened to pick a String/Integer mismatch, you will get syntax errors. Integer/Real mismatches are very difficult to locate.

Another peculiarity of trailing characters is that if you have a function named XYZ!, and you attempt to reference that function as XYZ# or XYZ% or anything other than XYZ! or XYZ then you will get a MULTIPLE DEFINITION error at the function declaration even though the reference is many lines later.

Use of Colons as Statement Separators

Use of colon ":" as a statement separator can cause a number of problems with both QuickBASIC and A-B BASIC. An example follows:

```
CL2 : PRINT "This is a test" ;
```

The programmer assumed that this statement would clear the second line of the display and then print “This is a test”. But, this is not what actually happens! This is compiled as:

```
CL2 :          PRINT “This is a test” ;
```

where CL2 is a local label which is effectively ignored.

The rule should be NEVER have a line which looks like:

```
< identifier > : < rest of line >
```

unless you INTEND for <identifier> to be a label.

CONSTant Declarations

“CONST name = value” does not quite follow the same rules as variable assignments in the same position. The major difference is that the type of ‘name’ is determined by the contents of ‘value’. A DEFINT or DEFSNG #, or $ after the name to set the type explicitly. The trailing symbol is NOT required when the constant is referenced.

PRINT “text”; : <statement>

A problem for Pascal and C programmers is coding: PRINT “text”, <statement> instead of: PRINT “text”; : <statement>

If the statement is a function such as “e = ReadStr%(3,s$)” then the result is for e to retain its previous value, and for an extra 0 or 1 to follow the printed text.

Appendix B

Differences Between A-B BASIC and QuickBASIC

(All references to “QB” refer to QuickBASIC. Any time the “QB manual” is mentioned, we are referring to the *QuickBASIC Language Reference Manual*.

Restrictions

All arrays *MUST* be DIMensioned.

STATIC and DYNAMIC arrays and metacommands *DO NOT* apply in A-B BASIC.

Periods are not allowed within a variable name, unless that variable is part of a user defined TYPE.

Double precision reals are treated as SINGLE precision.

The entire A-B BASIC program (i.e. the input to LXB), *MUST* reside in one file.

The exponentiation operator (^), may have a positive integer exponent only.

“@” AND “[“ AND ”]” = *NOT* valid outside “literals”.

Some of the general syntax implemented by some BASIC compilers is *NOT* allowed. The following syntax may be familiar to some users but is *NOT* mentioned in the QB manual and is *NOT* recognized by the A-B BASIC cross-compiler (LXB):

[and] instead of (and) to delimit array subscripts.

? as a synonym for PRINT.

PRINT#, INPUT#, and WRITE# without a space preceding #.

DOUBLE Precision Floating-Point Data

As a whole, A-B BASIC does *NOT* support double precision real arithmetic. Specifically, the interpreter in the workstation is NOT programmed to do DOUBLE math.

However, as a convenience, the A-B BASIC cross-compiler accepts the following DOUBLE-related syntax in most respects:

DEFDBL

suffix wherever valid.

Floating-point constants with 8 or more significant digits.

AS DOUBLE clauses where valid.

Functions involving DOUBLE such as CVD.

Nonetheless, remember that all floating-point calculations in the workstation are performed in SINGLE precision.

Restriction on the Exponentiation Operator

A-B BASIC supports exponentiation to positive integer powers only.

For example:

SUPPORTED: X ^ 2
NOT SUPPORTED: X ^ 5.7 or X 6 –2

In addition, exponentiation is implemented by repetitive multiplication, so small exponents are recommended.

Full real exponentiation is provided in the function called “Power”! in the BASIC source file MATH.BAS.

Unsupported Math Intrinsics

These intrinsic functions, relating to exponent and trigonometric functions are *NOT* implemented in the interpreter running in the workstation: ATN COS EXP LOG SIN SQR TAN

However, BASIC source code implementing all of the above is provided in the file MATH.BAS and may be copied into programs as required.

A-B BASIC Keywords

ABS (numeric-expression)

Returns the absolute value of a numeric expression. ABS(–1) and ABS(1) are both 1.

ASC (stringexpression)

Returns a numeric value that is the ASCII code for the first character in a string expression. CharVal = ASC(A$)

BEEP

Sounds the speaker. Produces the same sound as PRINT CHR$(7).

CALL name[(argumenlist)]

name

The name, limited to 40 characters, of the BASIC SUB being called. The name must appear in a SUB statement in the same module.

argumentlist

The variables or constants passed to the subprogram. Arguments in the list are separated by commas. Arguments passed by reference can be changed by the subprogram.

```
DO
    Message$ = “Please enter now.”
    CALL ShowPrompt (Message$)
    ‘ Same results are obtained by the following:
    CALL ShowPrompt (“Please enter now”)
LOOP

SUB ShowPrompt (Prompt$)
    LOCATE 1, 1, 0
    PRINT Prompt$;
    BEEP: SLEEP 3
END SUB
```

CDBL(numeric-expression)

Converts a numeric-expression to a double-precision number. Note the restrictions on DOUBLE precision usage above.

CHAIN filespec

Transfers control from current program to another program residing in file memory. COMMON is NOT supported. Therefore, all variables are lost between the CHAIN unless first stored in a file and retrieved after the CHAINing.

```
LINE INPUT Request$
IF Request$ = “JUMP” THEN CHAIN “JUMP”
```

A-B BASIC KEYWORDS (cont'd)

The interpreter does not tolerate a CHAIN to a non-existent program. Make sure that you have downloaded any program which will be CHAINed to, before any other program attempts to CHAIN to it, or the program will lock up, and the workstation will have to be re-booted! A special set of Network Directives is required to download a program for the express purpose of a subsequent CHAIN. Refer to Chapter 8 for a further discussion of these Network Directives.

CHR$(code)

Returns a one-character string whose ASCII code is the argument.

```
Msg$ = CHR$ (34) = "Quoted string" = CHR$ (34)
```

This example adds a double-quote character to the beginning and the end of the string.

CINT(numeric-expression)

Converts a numeric-expression to an integer by rounding the expression's fractional part.

CLNG(numeric-expression)

Converts a numeric-expression to a long (4-byte)integer by rounding the expression's fractioinal part.

CLOSE [[#]filenumber[,[#]filenumber]...]

Concludes I/O to a file or device. Closing some reserved files, such as HOT, have additional side effects. See the description of those devices for details.

```
OPEN "EMPLOYEE" FOR OUTPUT AS #1
PRINT #1, "Employee –"; EmpBadge$
CLOSE #1
```

CLS [{ 0 | 1 | 2 }]

Clears the front panel LCD, homes the cursor. All forms shown are accepted, but all have the same effect.

CONST constname = expression

In a procedure a CONST overrides a GLOBAL variable of the same name. The text of a string CONST is to be present in one place, regardless of the number of times it is referenced. However, because string CONST statements produce executable code, they should be placed at the start of a program.

```
CONST FALSE = 0, TRUE = NOT FALSE
```

A-B BASIC KEYWORDS (cont'd)

CSNG(numeric-expression)

Converts a numeric expression to a single precision value.

```
Row = CSRLIN
```

CVI, CVS, CVL, CVD

See QuickBASIC Language Reference Manual for specific information. TYPE...END TYPE is much more versatile for use with random files.

DATE$

Returns a string containing the current date or sets the date. The format of the date string is yymmddw. Where yy is the year, mm is the month, dd is the day, and w is the day of the week (1 = SUNDAY). Note that A-B BASIC will automatically compute and update the day of the week if it is omitted or specified incorrectly.

```
DateString$ – DATE$
```

DECLARE {FUNCTION | SUB}name(parameters)(BASIC Procedures)

Declares references to BASIC procedures and invokes argument type checking. Sometimes the QuickBASIC editor insists on adding a clause reading AS ANY. This syntax is tolerated but is unable to perform full parameter verification. SEE "prototyping".

```
DECLARE FUNCTION GetTime$ ( )
DECLARE SUB SetTime$ ( )
```

Procedure prototyping by A-B BASIC

The generic name for declaring a procedure's name and its parameters is "prototyping". LXB creates a prototype for every procedure encountered whether by the DECLARE statement, the FUNCTION statement, or the SUB statement.

To allow the user to include files of DECLARE statements without regard to whether the code for each procedure is also included. LXB will *NOT* issue an error if a procedure is DECLAREd but never defined. Multiple DECLAREs of the same procedure are permitted by should be identical. Use of a procedure without its definition *WILL* cause an error.

A-B BASIC KEYWORDS (cont'd)

Note: If prototypes are not present in the source when QuickBASIC run a program, it simply creates them. However, if they are missing when the LXB compiler is invoked, a compiler error results.

DEFtype

See QuickBASIC Language Reference Manual for specific details.

DEFINT letterrange[, letterrange]...
DEFSNG letterrange[, letterrange]...
DEFDBL letterrange[, letterrange]...
DEFLNG letterrange[, letterrange]...
DEFSTR letterrange[, letterrange]...

DIM [SHARED] variable[(subscripts)][AS type]...
Declares a variable and allocates storage space.

Note: QB will not allow a fixed length string (nor an array of them) to be passed to a procedure, neither will A-B BASIC. So, as an alternative, TYPE...END TYPE a variable or array of fixed length strings, and pass it instead.

OPTION BASE is *NOT* supported, so in order to set the lower subscript use the following: DIM A(lower TO upper) if zero is not preferred as the lowest subscript.

A declaration using AS precludes variables of the same name, even with type suffixes. For example, NumOfBytes! = 10 will cause an error if there was a preceding.

```
DIM NumOfBytes AS INTEGER

DIM SHARED ON$, OFF$, Now AS INTEGER
```

DO...LOOP [{WHILE/UNTIL} booleanexpression]

Repeats a block of statements WHILE a condition is TRUE or UNTIL a condition becomes TRUE.

```
DO: X = X + 1: LOOP WHILE X  200
```

END [{FUNCTION/IF/SELECT/SUB/TYPE}]

Ends a BASIC program, procedure, or block. See QuickBASIC Language Reference Manual for detailed information.

EOF(filenumber)

Tests for the end of file condition. See the chapter on Special Devices for details about specific devices . . .

A-B BASIC KEYWORDS (cont'd)

Effect of EOF on a sequential file: TRUE is returned whenever the file position is = file length. (End-of-File is reached)

Effect of EOF on binary or random files: TRUE is returned if the previous GET did not get "all" the data that was requested. For random reads, "all" means the number of bytes specified as the LENgth in the OPEN, or the variable's size, whichever is less.

For binary reads, "all" means the size of the variable.

The difference between binary/random and sequential mode is that, in the former, EOF returns FALSE even though the next GET might not return any data. Therefore, for binary and random files, do a GET but then test EOF. If TRUE is returned, the variable does not contain a full record and may not contain anything at all. Subsequent calls to EOF will continue to return TRUE. EOF "latches" in this state until a SEEK operation is performed. Some statements also reset this latch because they do an implied SEEK as part of their operation. OPEN is an example.

```
OPEN "EMPLOYEE" FOR INPUT AS #1
DO WHILE NOT EOF (1)
      LINE INPUT #1, EmpBadge$
      IF EmpBadge$ = BadgeInput$ THEN
            Found = TRUE
            EXIT DO
      END IF
LOOP
CLOSE # 1
```

EXIT {DO/FOR/FUNCTION/SUB}

Exits a DO...LOOP or FOR...NEXT loop, FUNCTION, or SUB. See QuickBASIC Language Reference Manual for detailed information. See previous entry for example.

FIX(x)

Returns the truncated integer part of x.

```
FOR counter = start TO end [STEP increment]
```

NEXT

Repeats a group of instructions a specified number of times.

A-B BASIC KEYWORDS (cont'd)

Note: The data type of the counter controls the precision of the start, end and increment expressions.

WRONG: FOR J% = 1 to 80000: statements: NEXT J%
RIGHT: FOR J& = 1 TO 80000: statements: NEXT J&

FRE(numeric expression)
FRE(stringexpression) < —NOT VALID
Returns the available memory.

When the numeric expression evaluates to 0, an INTEGER is returned representing the number of bytes remaining for program pseudo-code, variable data, and the program's pseudo stack.

When the numeric expression evaluates to 1, a LONG is returned representing the number of bytes which can be used by files. (Because files are allocated in memory by blocks, rather than a byte at a time, this value wilal decrement by some multiple of 256.)

```
ProgMem% = FRE (0)
FileMem&  = FRE (1)
```

FREEFILE
Returns the next free BASIC file number.

FREEFILE returns 0 when all file numbers are "in use", i.e. are assigned to open files or devices. A-B BASIC and QuickBASIC may not return the name file numbers, even in identical programs.

Note: Just because a file number is unassigned does not imply that a new file can be created. There is a maximum number of files which can exist, open or not. See the Limits section for specifics on A-B BASIC file and device limitations.

```
IF NumFilesOpen 10 THEN NextFileNum = FREEFILE
```

FUNCTION name[(parameterlist)][STATIC]
Declares the name, the parameters, and the code that form the body of a FUNCTION. The appearance of a FUNCTION's name on the right side of an = or in an argument list, is considered to be a call to that function, with or without an "(argument list)". Do not use FUNCTION procedures that perform I/O in I/O statements.

```
CurrTime$ = GetTime$

FUNCTION GetTime$ ( )
    GetTime$ = TIME$
END FUNCTION
```

A-B BASIC KEYWORDS (cont'd)

GET [#]filenumber[,[recordnumber][,variable]]

Reads data from a file in memory into a variable. All 3 items after GET are required, however the record number can be consecutive commas as in, GET #1,,V$. See the chapter on SpecialDevices for details about input from specific devices. FIELD is not supported, use TYPE...END TYPE.

```
TYPE EmpType
    Badge AS STRING * 8
    Name AS STRING * 25
END TYPE

DIM SHARED Emp AS EmpType
OPEN "EMPLOYEE" FOR RANDOM AS #1 LEN = LEN(Emp)

Rec = 0
DO WHILE NOT EOF(1)
    Rec = Rec + 1
    GET #1, Rec, Emp
    IF Emp.Badge = BadgeInput$ THEN Found = TRUE: EXIT DO
LOOP
CLOSE #1
```

GOSUB {linelabel | linenumber}....RETURN

Branches to and returns from a sub-routine. Only a RETURN is supported, not a RETURN linelabel. BASIC's SUB and FUNCTION procedures provide a more well-structured alternative to GOSUB....RETURN subroutines.

```
DO
CurrTime$ = LEFT$ (GetTime$,4)
    IF CurrTime$  OldTime$ THEN
        GOSUB UpdateTime
        OldTime$ = CurrTime$
    END IF
LOOP
UpdateTime:
LOCATE 1, 1, 0
PRINT LEFT$(CurrTime$,2); " : "; RIGHT$ (CurrTime$,2);
RETURN
```

GOTO {linelabel/linenumber}

Branches unconditionally to the specified line. See QuickBASIC Language Reference Manual for detailed information.

HEX$(expression)

Returns a string that represents the hexadecimal argument expression.

See QuickBASIC Language Reference Manual for detailed information.

A-B BASIC KEYWORDS (cont'd)

IF...THEN...ELSE

Allows conditionial execution, based on the evaluation of a Boolean expression.

See QuickBASIC Language Reference Manual for detailed information.

```
IF (A  B) AND (C  D)  THEN
    Sum = (A – B) + C
ELSEIF (A  B) AND (C  D)  THEN
    Sum = (B – A) * D
    ELSE
    LSE
    Sum = A * C
END IF
```

INKEY$

Reads a character from the keyboard.

Note: All key codes in a workstation are a single byte. The codes corresponding to special keys can vary due to differences in keypad configurations. If necessary, use this function to get a keystroke, convert it using ASC, then print the result.

```
A$ = " "
DO WHILE A$ = " " : A$ = INKEY$: LOOP
```

INPUT$(n[,[#]filenumber])

Returns a string read from the specified file. A common error is switching the byte count and the filenumber. To prevent this, always use the # symbol with the

```
J$ = INPUT$ (LOC(1), #1)
```

INSTR([start,]stringexpression1,stringexpression2)

Returns the character position of the first occurrence of a string in another string. See QuickBASIC Language Reference Manual for detailed information.

```
i = INSTR(DataIn$, CHR$(13)          ' Look for a carriage return.
```

INT(numeric-expression)

Returns the largest integer less than or equal to numeric-expression. See QuickBASIC Language Reference Manual for detailed information.

IOCTL$([#]filenumber)

Remarks 1 and 2 in the QuickBASIC Language Reference Manual are not applicable to A-B BASIC.

A-B BASIC KEYWORDS (cont'd)

```
OPEN "BAR" FOR INPUT AS #1
DO
    CLS
    PRINT "SCAN BADGE"
    DO WHILE NOT EOF(1) : LOOP      ' Wait for data...
    SOUND 1400,2
    LINE INPUT #1, BadgeInput$
    DataInfo$ = IOCTL$(1)
    Delimiter = INSTR(DataInfo$, " / ")
    DataType$ = LEFT$(DataInfo$, Delimiter – 1)
    DataSrce$ = MID$(DataInfo$, Delimiter + 1)
    IF DataType$ = "C39" AND DataSrce$ = "SLOT" THEN EXIT DO
LOOP
```

IOCTL[#]filenumber,string

See IOCTL$ statement. See the chapter on Special Devices for details about specific devices.

```
OPEN "HOST" FOR OUTPUT AS #1
IOCTL #1, "33"
```

Changes the last two digits of the ten digit header sent by the workstation to the HOST to 33.

KILL "filespec"

Deletes file "filespec" from memory and releases that memory space for use by future files.

KILL "!" deletes all files (except the QUE) in the workstation. "*.*" wildcards are not used to avoid problems with deleting files on the PC. ALL REMARKS in QB manual are applicable to A-B BASIC. Like QB, KILL is only valid on a CLOSEd file. However, A-B BASIC will not issue an error message if this is attempted. The QUE can be forcibly emptied by issuing a KILL "QUE" statement explicitly.

LBOUND(array[,dimension])

Returns the lower bound (smallest available subscript) for the indicated dimension of an array. The default lower bound is zero. Arrays dimensioned using the TO clause in the DIM statement may have any integer value as a lowerbound. Compliment of UBOUND.

LCASE$(stringexpression)

Returns a string expression with all letters in lowercase.

```
A$ = LCASES$( "NOW IS THE TIME..." )
```

A-B BASIC KEYWORDS (cont'd)

LEFT$(stringexpression, n)

Returns a string consisting of the leftmost n characters of a string. See Also MID$, and RIGHT$. See QuickBASIC Language Reference Manual for detailed information.

```
CurrTime$ = LEFT$ (TIME$,4)
    'Returns the leftmost 4 characters of the time.
```

LEN(stringexpression)
LEN(variable)

Returns the number of characters in a string or the number of bytes required by a variable. Refer to QuickBASIC Language Reference Manual for detailed information.

LINE INPUT[;]["promptstring";]stringvariable

Inputs an entire line (up to 255 characters) to a string variable, without the use of delimiters. The only editing character that is supported for LINE INPUT is a backspace. The files READ.BAS and MENU.BAS provide greatly enhanced input capabilities.

```
LINE INPUT "Enter name:  "; Name$
```

LINE INPUT #filenumber,stringvariable

Reads an entire line from a sequential file to a string variable. See the chapter on Special Devices for details about specific devices. See example listed for EOF().

```
OPEN "DATA.DAT" FOR INPUT AS #1
LINE INPUT #1, Title$
```

LOC(filenumber)

Returns the current position within a file. The second paragraph in QB manual under REMARKS does *NOT* apply to A-B BASIC. See the chapter on Special Devices for details about specific devices.

```
OPEN "COM" FOR INPUT AS #1          ' Must be a normal workstation.
DO
    DO WHILE LOC(1) = 0: LOOP
    R$ = INPUT$(1, #1)
    Rx$ = Rx$ + R$
    i = INSTR(Rx$, CHR$(13)         ' Look for a carriage return.
    IF i  0 THEN
        Detain$ = LEFT$(Rx$, i – 1)
        CLS
        PRINT Datain$;
        Rx$ = " ": R$ = " "
    END IF
LOOP
```

A-B BASIC KEYWORDS (cont'd)

LOCATE[row][,[column][,[cursor]]]

Moves the cursor to the specified position. The start and stop arguments do not apply to A-B BASIC.

```
LOCATE 1, 1, 0
```

LOF(filenumber)

Returns the length of the named file in bytes. The second paragraph in the QB manual under REMARKS does *NOT* apply to A-B BASIC. See the chapter on Special Devices for details about specific devices.

```
OPEN "POWER" FOR INPUT AS #1
IF LOF(1)  0 THEN PowerFail = TRUE
```

LTRIM$(stringexpression)

Returns a copy of a string with the leading spaces removed. See QuickBASIC Language Reference Manual for detailed information.

```
A = 100
A$ = LTRIM$ (STR$ (A))
        ' Removes leading space in front of 100.
```

MID$(stringexpression, start[, length])

Returns a substring of a string. See QuickBASIC Language Reference Manual for detailed information. See also MID$ statement, (replaces a portion of a string variable with another string.)

```
A$ = "Now is the time for..."
B$ = MID$ (A$, 8, 3)              ' B now equals "the".
```

MKD$, MKI$, MKL$, MKS$

Converts numeric values to string values. See QuickBASIC Language Reference Manual for detailed information.

OCT$(numeric-expression)

Returns a string representing the octal value of the numeric argument. See also HEX$().

ON expression GOSUB {line-number-list/line-label-list}
ON expression GOTO {line-number-list/line-label-list}

Branches to one of several specified lines, depending on the value of an expression. See QuickBASIC Language Reference Manual for detailed information.

```
N = 3
ON N GOSUB GetTime, GetDate, GetLost        ' Will gosub GetLost.
```

A-B BASIC KEYWORDS (cont'd)

OPEN file [FOR model][ACCESS] AS[#]filenum [LEN = reclen]

Enables I/O to a file or device. In QB manual Syntax 2 does not apply to A-B BASIC. The access argument syntax is accepted but is not enforced at run-time. The lock clause is not supported. See the chapter on Special Devices for details on OPENing specific devices.

```
TYPE EmpType
    Badge AS STRING * 8
    Name AS STRING * 25
END TYPE
DIM SHARED Emp  EmpType
OPEN "EMPLOYEE" FOR RANDOM AS #1 LEN = LEN(Emp)
```

POS(0)

Returns the current horizontal position of the cursor on the LCD display. A single argument is required but is ignored at run-time.

```
Col = POS(0)
```

PRINT [expressionlist][{, | ;}]

Outputs data to the LCD display. The format for some values may vary from QuickBASIC to A-B BASIC. A-B BASIC divides the line into print zones of 8 spaces each. A semicolon at the end of the print statement keeps the cursor from advancing to the next line.

```
PRINT "Enter your name: ";
LINE INPUT; Name$
```

PRINT #filenumber,[USING stringexpression;] expressionlist[{, | ;}]

Writes data to a file in memory. A print field width is 8 characters for the LCD, 14 for others. See Chapter 12 (Special Devices) for details on specific devices.

```
OPEN "EMPLOYEE" FOR APPEND AS #1
format$ = "$###.##"
A = 123.459
PRINT #1, USING Format$; A              ' Prints A to file #1 as
$123.45
```

PRINT USING formatstring; expressionlist[{, | ;}]

Prints strings or numbers using specified format. A field width is 8 characters for the LCD. INTEGERs and LONGs will output with .00 even if .## is used. If a number is negative, a minus sign(–) will always precede it. The following format string characters are not supported... **, $$, **$, ",". See the limits page in Appendix N for the maximum USING clause and the maximum output of a single PRINT.

```
Format$ = "$####.##"
A = 123.459
PRINT USING Format$; A              ' Prints A as $123.45
```

A-B BASIC KEYWORDS (cont'd)

PUT [#]filenumber[,[recordnumber][,variable]]

Writes from a variable or a buffer to a random or binary file. The required syntax is shown above. Omitting the recordnumber such as PUT #1,,variable is supported, but omitting both the record number and the variable is not supported. See Chapter 12 (Special Devices) for details on specific devices.

```
TYPE EmpType
    Badge AS STRING * 8
    Name AS STRING * 25
END TYPE
DIM SHARED Emp AS EmpType
OPEN "EMPLOYEE" FOR RANDOM AS #1 LEN = LEN(Emp)
Rec = Rec + 1
Emp.Badge = "1234"
Emp.Name = "Johnathan Q. Doe"
PUT #1, Rec, Emp
CLOSE #1
```

RANDOMIZE expression

Initializes (re-seeds) the random number generator. The expression is required and must be numeric.

```
RANDOMIZE TIMIER            ' Re-seeds random number generator.
```

REM remark
' remark

Allows explanatory remarks to be inserted in a program. Both syntax are supported.

RETURN

Returns control from a sub routine. NO return line number or line label is supported. ONLY RETURN.

RIGHT$(stringexpression, n)

Returns the rightmost n characters of a string.

```
A$ = "Hello, how are you? Have you been alright?
Through all... "
B$ = RIGHT$(A$, 14)            ' B$ = "Through all... "
```

RND

Returns a single-precision random number between 0 and 1. *NO* argument is to be placed behind RND. To produce random integers in a given range, use the following formula:

```
INT( (upperrange – lowerrange + 1) * RND + lowerrange)
```

A-B BASIC KEYWORDS (cont'd)

RTRIM$(stringexpression)

Returns a string with trailing (right-hand) spaces removed.

```
A$ = "When I was younger, so much younger than today          "
A$ = RTRIM$(A$)                ' Strips off trailing spaces.
```

SEEK(filenumber)

Returns the current file position. See the chapter on Special Devices for details on specific devices. See QuickBASIC Language Reference Manual for detailed information.

SEEK [#]filenumber,position

Sets the position in a file for the next read or write. See the chapter on Special Devices for details on specific devices.

```
SELECT CASE text expression
    CASE expressionlist1
        statements
    CASE expressionlist2
        statements
    CASE ELSE
        statements
END SELECT
```

Executes one or more CASE statements depending on the value of text expression. CASE ELSE is required in A-B BASIC. See QuickBASIC Language Reference Manual for specific details.

SGN(numeric-expression)

Indicates the sign of a numeric expression. See QuickBASIC Language Reference Manual for specific details.

```
SHARED variable [AS type] [,variable [AS type] ]...
```

Gives a SUB or FUNCTION procedure access to variables declared at the module level without passing them as parameters.

All arrays in the SHARED statement must have been previously DIMensioned in module level code. LXB also requires non-arrays to have previously appeared at the module level.

```
' Module level code...
DIM Datain AS STRING
DO
    LINE INPUT; Datain
    CALL ShowDatain
LOOP

' Sub-routine
SUB ShowDatain
SHARED Datain AS STRING
    LOCATE 1, 1, 0
    PRINT Datain
    END SUB
```

A-B BASIC KEYWORDS (cont'd)

SLEEP seconds

Suspends execution of the calling program. In the QB manual the first event under remarks is the only one supported.

```
SLEEP 5     ' Suspends execution for 5 seconds.
```

In A-B BASIC, fractional second delays are supported to a resolution of 1/100ths of a second. This is different from QuickBASIC, where delays under .5 are rounded down to zero, suspending program execution.

SOUND frequency,duration

Generates sound through the speaker. In the workstation, the correspondence between the "frequency" specification and the true sound of that frequency is approximate. In the workstation, (unlike the PC), the sound is initiated and the program continues without waiting for the sound to complete.

```
SOUND 1400, 2          ' 1400 Hz for 2 clock ticks.
                       ' 20 ticks per second.
```

SPACE$(n)

Returns a string of spaces of length n.

```
X = 25
A$ = SPACE$(X)         ' A$ equals 25 spaces.
```

STATIC variablelist

Makes simple variables or arrays local to either a FUNCTION, or a SUB and preserves values between calls. See QuickBASIC Language Reference Manual for detailed information.

For arrays, STATIC must precede DIM. Also any AS clauses must match. Also a single variable in STATIC may NOT also appear in a DIN.

STOP

Terminates the program. In the QB manual the 1st and 2nd paragraphs do not apply to A-B BASIC. Also see SYSTEM.

STR$(numeric-expression)

Returns a string representation of the value of a numeric expression. The format of the strings returned by A-B BASIC may not be identical to those returned by QuickBASIC. For example, given STR$(1.0), QB will return 1, but A-B BASIC will return 1.000000.

```
X% = 1234
S$ = STR$(X%)          ' S$ equals "  1234"
```

A-B BASIC KEYWORDS (cont'd)

STRING$(m,n)
STRING$(m,stringexpression)

Returns a string whose characters all have a given ASCII code or whose characters are all the first character of a string expression. See QuickBASIC Language Reference Manual for specific details.

```
Header$ = STRING$ (80, " * " )          ' Header$ equals 80 asterisks.
```

SUB globalname[(parameterlist)][STATIC]
...
...
END SUB

Marks the beginning and end of a subprogram. See also STATIC.

```
SUB GetName (Name$)
    DO
        CLS
        LINE INPUT "Enter your name:  ";Name$
        CLS
        PRINT Name$
        PRINT "Correct?  ";
        LINE INPUT; YN$
        YN$ = UCASE$ (YN$)
        IF YN$ = "Y" OR YN$ = "  " THEN EXIT DO
    LOOP
END SUB
```

SWAP vari1,vari2

Exchanges the values of two variables. See QuickBASIC Language Reference Manual for specific details.

```
A$ = "1234
B$ = "4321"
SWAP A$, B$          ' A$ = "4321", B$ = "1234"
```

SYSTEM

Terminates program execution. END, STOP, and SYSTEM as well as "reaching the bottom" of a A-B BASIC program all have the same result: the program is terminated. Program termination causes the following:

1. Files are left unchanged. In particular, if HOST was OPEN, it remains OPEN.
2. The front panel LCD shows the status display. The BASIC run time status display and the corresponding read-only menu display show "End by System".
3. An error: "BASIC ERROR: End by System" is placed on the network for transmission to the HOST.
4. "End by System" with an offset of 0 (zero) is serious and should be reported.

A-B BASIC KEYWORDS (cont'd)

TIME$

Returns the current time from the operating system. An 8 byte string is returned in the form hhmmssnn, representing hours (0-23), minutes, seconds, and hundredths of a second. (The Clock Mode setting of the Setup Menus has no effect on the format of the TIME$ variable.)

```
CurrTime$ = TIME$
```

TIME$ = stringexpression

Sets the time. See preceding entry.

```
TIME$ = "15450000"      ' Sets time to 3:45:00.00
```

TIMER

Returns the number of seconds elapsed since midnight. As a single precision value this is accurate to 1/100th of a second. Measurements with this functioin can be in error if they span midnight. See also the EGG device.

```
StartTime = TIMER
FOR X = 1 TO 500:  NEXT X
EndTime = TIMER
Elapsed = EndTime – StartTime
```

TYPE usertype
elementname AS typename
elementname AS typename
END TYPE

Defines a data type containing one or more elements. An element cannot be the same of an array.

```
TYPE EmpType
    Badge       AS STRING * 8
    Name        AS STRING * 25
    CodeNum     AS INTEGER
END TYPE
```

UBOUND(array[,dimension])

Returns the upper bound (largest available subscript) for the indicated dimension of an array. See QuickBASIC Language Reference Manual for specific details.

UCASE(stringexpression)

Returns a string expression with all letters in uppercase.

```
A$ = UCASE$( "Now is the time for all..." )
A$ now contains:  NOW IS THE TIME FOR ALL...
```

A-B BASIC KEYWORDS (cont'd)

VAL(stringexpression)

Returns the numeric value of a string of digits. See QuickBASIC Language Reference Manual for specific details.

```
Amount = VAL(CheckAmt$)
```

WHILE condition
...[statements]
WEND

Executes a series of statements in a loop, as long as a given condition is true. A-B BASIC does impose a nested loop limit which should not be exceeded under normal programming conditions.

```
WHILE INKEY$ = "  ": WEND          ' Loops until a key is pressed.
```

WRITE #filenumber,expressionlist

Writes data to a file in memory. See the chapter on Special Devices for details on specific devices.

Although the syntax specification (as printed) indicates that the "expressionlist" is required, the text states and both QB and A-B BASIC compilers treat it as optional. The syntax is WRITE#n,[expression].

Appendix C

Application Library Subroutines

In ENV.BAS and ENVPC.BAS

OpenLINX . . . . . Opens all devices to be used by low level I/O routines
TestEvent% . . . . . Tests for an event
Read Event% . . . . Waits for an event
Send . . . . . Sends a record directly to the host
SendQue . . . . . Sends a record indirectly to the host through the queue
SetLINE . . . . . Sets the state of the control line
SetLED . . . . . Turns a workstation keyboard LED on or off
SendCom . . . . . Sends a record to the comm line
Set DTR . . . . . Sets the state of Data Terminal Ready
SetRTS . . . . . Sets the state of Request To Send for a comm port
GetDSR% . . . . . Gets Data Set Ready status for a comm port
PENDING% . . . . . TRUE if Send will result in a pause.
MEMFULL% . . . . TRUE if less than 1024 bytes remain in RAM
ONLINE% . . . . . TRUE when ONLINE to the network
GetDATE$ . . . . . Translates DATE$ into standard form (MM/DD/YYYY)
GetTIME$ . . . . . Translates TIME$ into standard form (HH:MM:SS)

In READ.BAS

CL1 . . . . . Clears the top line of the display and homes cursor
CL2 . . . . . Clears line 2 and left justifies cursor
PC1 . . . . . Prints a string centered on line 1
PC2 . . . . . Prints a string centered on line 2
ReadFlush . . . . . Flushes the keyboard buffer
ReadStrField% . . . Edit a string field on the display
ReadStr% . . . . . Simple edit a string on the screen
ReadNumSTR% . . Simple edit of string of numeric characters
ReadPassword% . . Inputs a password from the keyboard
ReadYN% . . . . . Wait for a YES/NO response from user
ReadInt% . . . . . Input an integer value from the keyboard
ReadReal% . . . . . Input a real value from the keyboard
ReadLong% . . . . . Input a long value from the keyboard

In MENU.BAS

Menu% . . . . . Displays a menu and waits for a selection
LogE! . . . . . Returns the natural log of a value
Log10! . . . . . Returns the base 10 log a value
Exponential! . . . . . Returns e to the x
Power! . . . . . Relturns x to the power of y
SquareRoot! . . . . . Returns the square root of x
CubeRoot! . . . . . Returns the cube root of x

In TRIG.BAS

FullArcTangent! . . Returns arc tangent using numerator and denominator
ArcTangent! . . . . . Relturns the inverse tangent in radians
ArcSine! . . . . . Returns the inverse sine in radians
ArcCosine! . . . . . Returns the inverse cosine in radians
Tangent! . . . . . Returns the tangent of an angle in radians
Sine! . . . . . Returns the sine of an angle in radians
Cosine! . . . . . Returns the cosine of an angle in radians

In HYP.BAS

TangentH . . . . . Hyperbolic tangent
SineH . . . . . Hyperbolic sine
CosineH . . . . . Hyperbolic cosine
ArcTangentH . . . . Returns inverse hyperbolic tangent
ArcSineH . . . . . Relturns inverse hyperbolic cosine for x = 0
ArcCosineH . . . . . Returns inverse hyperbolic cosine for principal values

OpenLINX Opens all devices to be used by low level I/O routines.

This MUST be called before any of the Event or Read routines. This performs the actual BASIC OPEN statements for all of the devices which are accessed by the low level I/O.

Note: The global constant PcMode will be TRUE if th is program is running using EnvPC.BAS instead of Env.BAS. You may use this for any operations which are not fully emulated by the EnvPC environment. This also establishes the global environment variables.

```
DSPWIDTH% = 40 or 16 depending on the type of workstation
TERMNUM% = The current terminal number
TERMTYPE$ = “NORML”, “MASTR”, etc

OpenLINX
SUB OpenLINX
```

TestEvent% Tests for a low level event returns 0 when no event has occurred. Events are either keys read from the workstation keyboard, or special codes. Special codes are always negative, although some special keys also return negative codes. See OpenLINX for initialization information.

The most important events are:

TimeoutEvent . . . . The input timer has expired
NetEvent . . . . . Read from network has occurred
ComEvent . . . . . Read from the main comm port
AuxEvent . . . . . Read from the aux comm port

When ComEvent or AuxEvent has occurred, InCom$ contains the LINE INPUT format line which was read from the comm line. When NetEvent has occurred, InNet$ contains the record from the host. The events which you want to test for are indicated by the "mask" parameter. Each device which you want tested MUST have a bit set in the mask. Bits are set by building a mask using the constants: KbdMask, TimeoutMask, BadgeMask, NetMask, ComMask, AuxMask, and SensorMask.

To simplify matters, some special mask values are also allowed. These are:

0 = Read from keyboard only
1 = Read from keyboard with timeout (see ReadEvent%)
2 = Read from keyboard and SLOT, etc. with timeout

Note: Timeouts only apply to reads which wait for an event. TestEvent% never waits. See ReadEvent% for more info.

```
e = TestEvent%   ( mask% )

FUNCTION TestEvent% ( mask% )
```

SaveEvent Saves the previous event to allow it to happen again.

Saves the event which has just occurred from TestEvent%. The very next TestEvent% will return this event code.

```
SaveEvent eventcode%
SUB SaveEvent   ( e% )
```

ReadEvent% Waits for a low level event.

Waits for an event to occur. The event(s) to wait for are specified in the same manner as TestEvent%.

For timeouts set the global variable TIMEOUT% to the number of 1/100 second intervals you want to wait before a TimeoutEvent is returned. The value of TIMEOUT% is not modified by %.

```
e = %   ( mask );
FUNCTION % ( mask% )
```

SetDTR Sets the state of Data Terminal Ready.

SetRTS Sets the state of Request To Send for a comm port.

Set state = TRUE to turn DTR or RTS on. You may examine the global variables RTS1%, DTR1%, RTS2%, and DTR2% to get the current state.

```
n% is 1 for COM and 2 for AUX.
SetDTR n%,  state%
SetRTS n%,  state%
SUB SetDTR  (  n%, s% )
SUB SetRTS  (  n%, s% )
```

GetDSR% Gets Data Ready status for a comm port.

A TRUE is returned if DSR is ON. n% is 1 for COM and 2 for AUX.

```
state = GetDSR%(n%)
FUNCTION etDSR% ( n% )
```

SendCom Sends a record to a comm line.

A string is sent to the comm line. The string is sent exactly as is, so append CR or CR and LF if you are sending a line. n% is 1 for COM and 2 for AUX.

Note that COMOPEN% must be true for a send to COM to work. COMOPEN% is true only if the TERMTYPE% is TypeNormal.

```
SendCom  n%,   s$
SUB SendCom  (  n%, s$ + CHR$(CR) )
```

ONLINE% TRUE if the workstation is online.

```
IF ONLINE% THEN . . .
```

MEMFULL% TRUE if less than 1024 bytes remain in RAM.

This should be checked before any SendQue or outputs to the RAM files to prevent memory full.

```
IF MEMFULL% THEN . . .
```

PENDING% TRUE if Send will result in a pause.

This should be checked before any Send, to prevent pauses in execution.

```
WHILE PENDING%
. . .

WEND
```

Send Sends a record directly to the host.

A string is sent to the host from this terminal. You should test ONLINE% and PENDING% if you don't want this to pause.

```
Send    s$
```

SendQue Sends a record indirectly to the host through the que.

A string is placed into the network queue. The record will in turn be sent to the host as soon as possible. SendQue has a lower priority than Send, and will buffer records even when NOT ONLINE.

This subroutine uses the "Host" device, *NOT* the "Que" device. See "Host Device" discussion.

Pause will occur if the unit was not ONLINE% and MEMFULL%.

```
SendQue    s$
```

SetLED Turns a workstation keyboard LED on or off.

A workstation has up to 10 LEDs. The LEDs are numbered 1 to 10. state% should be TRUE for ON and FALSE for OFF.

```
SetLED    n%,    state%
```

**GetTIME$** Translates TIME$ into standard form (HH:MM:SS).

In A-B Basic TIME$ is in the form hhmmsshh. This routine drops the hundreth seconds and puts colons between hours/minutes and minute/seconds.

```
GetTIME$
```

**GetDATE$** Translates DATE$ into standard form (MM/DD/YYYY).

In A-B Basic DATE$ is in the form yymmddx. This routine ignores the day (the x).

```
GetDATE$
```

Note: In EnvPC, dashes are returned, not slashes (MM-DD-YYYY).

CL1 Clears the top line of the display and homes cursor.

CL2 Clears line 2 and left justifies cursor.

```
SUB  CL1
SUB  CL2
```

PC1 Prints a string centered on line 1.

PC2 Prints a string centered on line 2.

```
PC1  s$
SUB PC1  (  s$  )
SUB PC2  (  s$  )
```

ReadFlush Flushes the keyboard buffer.

This clears the keyboard buffer of any pending keystrokes.

```
ReadFlush
SUB ReadFlush
```

ReadStrField% Input a string on the display

The string$ contents are placed on the screen and may be edited by the user. This default may be inhibited using NODEFAULT.

The attr% is made by adding the following:

AUTOEXIT . . . . . Causes the field to be auto exited
BLANKPAD . . . . . Selects blank padding instead of underscores
SECURE . . . . . The field is displayed as asterisks
NOEDIT . . . . . Displayed but cannot be edited
NOCLEAR . . . . . Inhibits auto clear on input
NOCURSOR . . . . Inhibits display of cursor during edit
FORMEXIT . . . . . Allow arrows to exit the field entry
NODEFAULT . . . . Do not use the default value in the string
USETABLE . . . . . Use CHARTABLE$ to restrict legal characters

The fieldlength% is in characters. The startingchar% specifies the offset from the beginning of the field at which editing is to start, this begins at 0. Note that the global EXITOFFSET% is set to the cursor position at exit. These may be used to restart a read which was interrupted by some event.

The global MODIFIED% is TRUE when the string was changed in any way. The strlength is the maximum number of characters to allow in the string (not including the terminal null). This must be at least fieldlength% characters long. The return value “e” is an event code. This is 0 on a normal return (operator pressed CR). The devices which are active are defined by the global variable GLOBALMASK% which is in the same format as the mask% used by % and TestEvent%. The field will start at the current cursor location. The cursor is left at its starting locationi when ReadStrField exits.

```
e = ReadStrField%(  attr%, fldlen%, strtchar%, string$,
stringlen%  )
FUNCTION ReadStrField% (  attr%, flength%, strpos%, s$,
slen%  )
```

ReadStr% Edit a string on the screen.

Thhis is a simplified version of ReadStrField% that does not have as many parameters. Any attributes should be in the GLOBALATTR% variable. Don’t forget to set GLOBALMASK% to the devices which can terminate the read. Make sure you include KbdMask. See TestEvent% for more info.

```
e = ReadStr  (  length%, s$  )
FUNCTION ReadStr%  (  n%, s$  )
```

An example of reading from the keyboard is as follows:

```
e = ReadStr(Length%,  s$)
IF e = BadgeEvent  THEN
 . .  s$ = InBuf$
END IF
```

ReadNumStr% Edit a numeric string on the screen.

The string which is edited may contain only 0 through 9. Any attributes should be in the GLOBALATTR% variable.

```
e = ReadNumStr  (  length%,  s$  )
FUNCTION ReadNumStr%  (  n%,  s$  )
```

ReadYN% Wait for a YES/NO response from user.

Returns TRUE or FALSE depending on the user’s response. EXITKEY% is set if you really need the event value. Nothing is displayed; this just waits for Enter, Clear, or a Y or N.

```
IF ReadYN% ( default% ) THEN . . .
FUNCTION ReadYN% ( defv% )
```

ReadPassword% Inputs a password from the keyboard.

```
e = ReadPassword%  (  length%, string$  ) ;
FUNCTION ReadPassword%  (  n%,  s$  )
```

ReadInt% Input an integer value from the keyboard.

To suppress display of the default value, use a negative length.

```
e = ReadInt% (  length%, int%  )
FUNCTION ReadInt% (  n%, v%  )
```

ReadLong% Input a long value from the keyboard.

To suppress display of the default value, use a negative length.

```
e = ReadLong% (  length%, long&  )
FUNCTION ReadLong% (  n%, v&  )
```

ReadReal% Input a real value from the keyboard.

The precision specifies how many decimal points to display. It does not affect the precision of the value input by the user. Use a negative length to suppress the default value.

This only handles floating point values. NOT scientific forms.

```
e = ReadReal% (  length%, precision%, real!  ) ;
FUNCTION ReadReal% (  n%, p%, v!  )
```

Menu% Displays a menu and waits for a selection.

A menu$ is provided which is used to format a menu on the two lines of the workstation display. The operator can select a menu entry by using 1 of 2 methods; these are:

1. Press a LETTER or NUMBER which corresponds to the first UPPER CASE letter or NUMBER of an entry.
2. Using the cursor keys to move to the item to be selected, then pressing Enter.

Regardless of the selection method used, Menu% will always return an event code of 1 through 10 for the item which was selected. If the operator pressed Clear or Exit, a functioin key, or an event occurred which was enabled by GLOBALMASK%, then these codes will be returned instead of 1 through 10. See Env.BAS for information on use of GLOBALMASK%.

The menu string is a string in which each menu entry is preceded by a " | " (vertical bar). Thus a menu$ such as:

"Payment Type: | Mastercard | Visa | Amex \ " + " | Other | Cash"

has 5 entries; "Mastercard" is the 1st entry and may also be selected by pressing M. "Cash" is the 4th entry and may also be selected by pressing C. Note the use of " \ " to start a new line (the fact that the string was made by adding two strings together cannot be detected by Menu%).

The actual menu displayed will look like:

```
Payment Type: <Mastercard>  Visa    Amex
                Other               Cash
```

The startingitem% is a value of 1 to 10, to indicate wh ich entry should be selected first. Note that the global EXITOFFSET% is set to the entry which was selected at exit.

```
e = Menu% ( startingitem%, menu$ )
FUNCTION Menu% ( sitem%, m$ )
```

LogE! Returns the natural log of a value.

```
x! = LogE! ( x! )
FUNCTION LogE! ( x )
```

Log10! Returns the base 10 log of a value.

```
x! – Log10! ( x! )
FUNCTION Log10! ( x )
```

Exponential! Returns e to the x

```
x! = Exponential! ( x! )
FUNCTION Exponential! ( x )
```

Power! Returns x to the power of y.

```
z! = Power! ( x!, y! )
FUNCTION Powerl! ( x, y )
```

CubeRoot! Returns the cube root of x.

```
x! = CubeRoot! ( x! )
FUNCTION CubeRoot! ( x )
```

SquareRoot! Returns the square root of x.

```
x! = SquareRoot! ( x! )
FUNCTION SquareRoot! ( x )
```

FullArcTangent! Returns arc tangent using numerator and denominator.

This is similar to ArcTangent (numerator/denominator) except it avoids the divide by 0 problems and sign problems of the normal ArcTangent function. The angle is returned in radians.

Example: Usually the numerator and denominator are y and x respectively. When x is zero, y/x is undefined, but the FullArdcTangent of y,x is pi/2 or pi/2 depending on the sign of y.

```
x! = FullArcTangent!  (  numerator!, denominator!  )
FUNCTION FullcTangent!  (  num, den  )
```

ArcTangent! Returns the inverse tangent in radians.

```
angle! = ArcTangent!  (  y!/x!  )
FUNCTION ArcTangent!  (  x  )
```

ArcSine! Returns the inverse sine in radians.

```
angle! = ArcSine!  (  x!  )
FUNCTION ArcSine!  (  x1  )
```

ArcCosine! Returns the inverse cosine in radians.

```
angle! = ArcCosine!  (  x!  )
FUNCTION ArcCosine!  (  x  )
```

Tangent! Returns the tangent of an angle in radians.

```
x! = Tangent!  (  angle!  )
FUNCTION Tangent!  (  x1  )
```

Sine! Returns the sine of an angle in radians.

```
y! = Sine!  (  angle!  )
FUNCTION Sine!  (  x1  )
```

Cosine! Returns the cosine of an angle in radians.

```
x! = Cosine!  (  angle!  )
FUNCTION Cosine!  (  x  )
```

TangentH! Returns the hyperbolic tangent.

```
y! = TangentH!  (  x!  )
FUNCTION TangentH!  (  x  )
```

SineH! Returns the hyperbolic sine.

```
y! = SineH! ( x! )
FUNCTION SineH! ( x )
```

CosineH! Returns the hyperbolic cosine.

```
y! = CosineH! ( x! )
FUNCTION CosineH! ( x )
```

ArcTangentH! Returns the inverse hyperbolic tangent.

```
y! = ArcTangentH! ( x! )
FUNCTION ArcTangentH! ( x )
```

ArcSineH! Returns the inverse hyperbolic cosine for x = 0.

```
y! = ArcSineH! ( x! )
FUNCTION ArcSineH! ( x )
```

ArcCosineH! Returns the inverse hyperbolic cosine for principal values.

```
y! = ArcCosineH! ( x! )
FUNCTION ArcCosineH! ( x )
```

Appendix D

A-B BASIC Limits

Firmware Version	1.1
Maximum PRINT USING clause	80 characters
Maximum line to PRINT	200 characters
User status display, each line	40 characters
Maximum barcode	80 characters
Maximum record to network	110 characters (system adds another 10 bytes)
Maximum string from INPUTS$	200 characters
Maximum string from LINE INPUT	200 characters
Maximum string length theoretically	32K, but limited by memory
Maximum RAM files, open or closed	10
Maximum number of files and devices simultaneously open	ALL files and all devices
Maximum file number	a value equal to the maximum number of files and devices = 24
Maximum size for program, variables, and temporaries	13700 bytes
Maximum individual RAM file size	Same as total RAM file size
Available memory for RAM files or programs to be CHAINed	Total Memory Size – 20K

Example: Maximum available memory in a 32K unit is 12K

Maximum recommended individual record size for a file	4K bytes
Maximum characters in a file name	12 characters

Other maximums, such as identifier length, are per QuickBASIC specification.

When compiling programs with the LXBC command, the statistics in the LST listing file show the percent of total compiler capacity used. Maximums can be calculated from the percentages shown.

How to compute memory size required for any application:

Interpreter overhead 6.3K
Program running 13.7K

# of CHAINed programs * 13.7K = . . . ??.? K
# of host records/hour
* Hours offline * bytes/rec = ... ??.? K
# of RAM file records for
lookup tables, etc. * bytes/rec = ... ??.? K

Total RAM Required = Sum of the above values

ALLEN-BRADLEY
A ROCKWELL INTERNATIONAL COMPANY

A subsidiary of Rockwell International, one of the world's largest technology companies, Allen-Bradley meets today's automation challenges with over 85 years of practical plant floor experience. More than 13,000 employees throughout the world design, manufacture and apply a wide range of control and automation products and supporting services to help our customers continuously improve quality, productivity and time to market. These products and services not only control individual machines, but also integrate the manufacturing process while providing access to vital plant floor data that can be used to support decision–making throughout the enterprise.

With offices in major cities worldwide.

WORLD HEADQUARTERS
Allen-Bradley
1201 South Second Street
Milwaukee, WI 53204 USA
Tel:(414) 382–2000
Telex:43 11 016
FAX:(414)382–4444

EUROPE/MIDDLE EAST/ AFRICA HEADQUARTERS
Allen–Bradley Europe B.V.
Amsterdamseweg 15
1422 AC Uithoorn
The Netherlands
Tel:(31) 2975/43500
Telex:(844) 18042
FAX:(31) 2975/60222

ASIA/PACIFIC HEADQUARTERS
Allen–Bradley (Hong Kong) Limited
Room 1006, Block B, Sea View Estate
2-8 Watson Road
Hong Kong
Tel:(852)887-4788
Telex:(780) 64347
FAX:(852)510-9436

CANADA HEADQUARTERS
Allen–Bradley Canada Limited
135 Dundas Street
Cambridge, Ontario N1R 5X1
Canada
Tel:(519)623–1810
FAX:(519)623–8930

LATIN AMERICA HEADQUARTERS
Allen-Bradley
1201 South Second Street
Milwaukee, WI 53204 USA
Tel:(414)382–2000
Telex:43 11 016
FAX:(414)382–2400

Catalog Number 2708–ND004 — November 1991

40062–227–01(A)
Printed in USA