ZBI

Zebra BASIC Interpreter

Command and Tutorial Reference

Zebra®

When it's on the line.

ZBI: Zebra BASIC Interpreter

Command and Tutorial Reference

Rev. 1
Customer Order #45543L
Manufacturer Part #45543LB

Proprietary Statement

This manual contains proprietary information of Zebra Technologies Corporation. It is intended solely for the information and use of parties operating and maintaining the equipment described herein. Such proprietary information may not be used, reproduced, or disclosed to any other parties for any other purpose without the expressed written permission of Zebra Technologies Corporation.

Product Improvements

Continuous improvement of products is a policy of Zebra Technologies Corporation. All specifications and signs are subject to change without notice.

Liability Disclaimer

Zebra Technologies Corporation takes steps to assure that its published Engineering Specifications and Manuals are correct; however, errors do occur. Zebra Technologies Corporation reserves the right to correct any such errors and disclaims liability resulting therefrom.

No Liability for Consequential Damage

In no event shall Zebra Technologies corporation or anyone else involved in the creation, production, or delivery of the accompanying product (including hardware and software) be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or the results of use of or inability to use such product, even if Zebra Technologies Corporation has been advised of the possibility of such damages. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

Copyrights

Table of Contents

SECTION ONE: Understanding the ZBI Environment

SECTION TWO: ZBI Programming Commands

SECTION ONE
Understanding the ZBI Environment

Overview of ZBI

The Zebra BASIC Interpreter (ZBI™) was designed specifically for interaction between the X.10 firmware environment and the ZPL II® Programming Language to create an even more powerful printing environment. ZBI allows you to maximize bar code printing options through custom programs written for a specific need.

To its advantage, ZBI bears a strong resemblance to many other BASIC interpreters and does not require you to learn an entirely new language from the ground up to take advantage of its features. For a complete listing of ZBI commands and terms, refer to Section Two on page 7. For additional information on ZPL II commands, refer to *ZPL Programming Guide Volume One* (45541L) and *Volume Two* (45542L).

In this section you will find several examples of how ZBI and ZPL II interact in bar code printing. It is recommended that you run several examples to gain an understanding of how the interpreter works in conjunction with your printer.

Starting the Zebra BASIC Interpreter

ZBI is enabled by sending a ZPL II command to the printer via one of the communication ports (serial, parallel, or Ethernet) in an interactive mode. This communication port is referred to as the console. Programs like HyperTerminal® for Windows™ can be used to communicate with your printer.

Two ZPL II commands are available to start the ZBI Interpreter: ~JI and ^JI. When the printer is turned on, it will accept ZPL II commands and label formats. *However, to receive ZBI commands the printer must be initialized using* ~JI *or* ^JI.

With a console application open, send one of these two commands to the printer:

1. **~JI** – when this command is received, the printer responds by sending a ZBI header with the program version and a new line containing an input prompt (>) back to the screen. This is an indication that ZBI program lines or commands may be sent to the printer. While receiving programming commands, the printer "echos" the received characters back to the source. This echoing can be turned off with the `ECHO` command.
2. **^JId:o.x,b,c,d** – sending this command to the printer will also activate a ZBI session. The parameters have the following functions:

d = location of program to run after initialization
Accepted Values: R:, E: and B:
Default Value: location must be specified

o = name of program to run after initialization
Accepted Values: any valid program name
Default Value: a name must be specified

x = extension of program to run after initialization
Fixed Value: .BAS

continued ►

b = console control
Accepted Values: Y (console on) or N (console off)
Default Value: Y

c = echoing control
Accepted Values: Y (echo on) or N (echo off)
Default Value: Y

d = memory allocation for ZBI
Accepted Values: 20K to 1024K
Default Value: 50K

If a second ~JI or ^JI command is received while the interpreter is running, the command will be ignored.

ZBI Interpreter Modes

The ZBI Interpreter operates in two modes:

1. The ***interactive mode*** allows commands to be entered and processed immediately. Command lines entered without line numbers are interactive and will be processed as soon as they are received by the printer.

   Example:

   ```
   >NEW
   ```

2. The ***program mode*** allows a series of command lines to be stored in memory and processed in numerical order when a RUN command is entered. Line numbers must be greater than 0 and less than 10,000.

   Example:

   ```
   >10 LET A=10
   >20 LET B=3
   ```

A program is activated from the first numbered line by entering the `RUN` command. While the program is running, it can be halted with the Control-C (^C) command, sent to the printer from the console. The interrupted program will continue where it left off by entering the `RESTART` command.

Ending A ZBI Session

An active ZBI session can be halted in one of two ways:

1. Sending ZPL at the input prompt (>).
2. Sending ~JQ at the input prompt (>).

ZBI Examples

Example 1

This example generates labels with a weight measurement that the user defines. After the file is loaded and activated, data is entered that will appear on the label.

```
5 ECHO OFF
10 CLOSE #1
20 CLOSE #0
30 OPEN #2 : NAME "SER"
40 OPEN #1 : NAME "ZPL"
50 DO
55 SLEEP 1
60 PRINT #2 : "W";
70 INPUT #2 : A$
80 IF A$ = "EXIT" THEN
90 CLOSE #2
100 OPEN #0 : NAME "SER"
105 END
110 END IF
120 LOOP WHILE POS(A$,"000.00") = 1 OR POS(A$,"?")=1
130 PRINT #1 : "~SD25^XA^MTD^FS^PW400^FS";
140 PRINT #1 : "^LH0,0^FS";
150 PRINT #1 : "^FO56,47^A0N,69,58^FDThis weighs^FS";
160 PRINT #1 : "^FO56,150^A0N,69,58^FD"&A$&" lbs^FS";
170 PRINT #1 : "^PQ1,0,0,N";
180 PRINT #1 : "^XZ"
190 DO
200 PRINT #2 : "W";
210 INPUT #2 : A$
220 LOOP UNTIL POS(A$,"000.00") = 1 OR POS(A$,"?")=1
230 GOTO 50
```

Example 2

Entering the code listed below will take an array that is five strings long and send the strings to the printer. In this example, the array must be declared.

```
10 OPEN 1# : NAME "ZPL"
20 DECLARE STRING MYARRAY$(5)
30 FOR INDEX = 1 TO 5 STEP 1
40 INPUT MYARRAY$(INDEX)
50 NEXT INDEX
60 PRINT #1 : "^XA^FO20,20^FD"&MYARRAY(1)&"^FS";
70 PRINT #1 : "^FO20,80^FD"&MYARRAY(2)&"^FS";
80 PRINT #1 : "^FO20,140^FD"&MYARRAY(3)&"^FS";
90 PRINT #1 : "^FO20,200^FD"&MYARRAY(4)&"^FS";
100 PRINT #1 : "FO20,260^FD"&MYARRAY(5)&"^FS^XZ"
```

Example 3

This is an example of using an AUTOEXEC file, and loading the START.BAS program from E: when the interpreter is activated. Note that the files START.BAS and AUTOEXEC.ZPL must already exist in their specified locations.

```
^XA^DFE:AUTOEXEC.ZPL^FS
^JIE:START.BAS,Y,N^FS
^XZ
```

Example 4

This example illustrates ZBI's data extraction capabilities. ZBI can be used to collect data from programs that are not ZPL-based. That information can then be imported into an existing ZPL label format that a Zebra printer will generate.

```
10 CLOSE #1
20 CLOSE #0
30 OPEN #2 : NAME "SER"
40 OPEN #1 : NAME "ZPL"
50 DO
60 INPUT #2 : A$
70 IF A$ = "EXIT" THEN
80 CLOSE #2
90 OPEN #0 : NAME "SER"
95 END
100 END IF
110 LOOP WHILE POS(A$,"START") = 0
120 DECLARE STRING BS$(17)
130 LET INDEX = 1
140 DO WHILE INDEX <= 17
150 INPUT #2 : B$
160 IF POS(B$,"DATA") <> 0 THEN
170 LET BS$(INDEX) = EXTRACT$(B$,",",";")
180 LET INDEX = INDEX + 1
190 END IF
200 LOOP
210 PRINT #1 : "~SD25^XA";
220 PRINT #1 : "^XFE:LABEL^FS";
230 PRINT #1 : "^FN1^FD"&BS$(2)&"^FS";
240 PRINT #1 : "^FN2^FD"&BS$(3)&"^FS";
250 PRINT #1 : "^FN3^FD"&BS$(4)&"^FS";
260 PRINT #1 : "^FN4^FD"&BS$(9)&"^FS";
270 PRINT #1 : "^FN5^FD"&BS$(10)&"^FS";
280 PRINT #1 : "^FN6^FD"&BS$(11)&"^FS";
290 PRINT #1 : "^FN7^FD>;>8"&BS$(15)&"^FS";
300 PRINT #1 : "^FN8^FD"&BS$(14)&"^FS";
310 PRINT #1 : "^FN9^FD>;>8"&BS$(16)&"^FS";
320 PRINT #1 : "^FN10^FD"&BS$(5)&"^FS";
330 PRINT #1 : "^FN11^FD"&BS$(6)&"^FS";
340 PRINT #1 : "^FN12^FD"&BS$(17)&"^FS";
350 PRINT #1 : "^FN13^FD"&BS$(13)&"^FS";
360 PRINT #1 : "^FN14^FD"&BS$(7)&"^FS";
370 PRINT #1 : "^FN15^FD"&BS$(12)&"^FS";
380 PRINT #1 : "^XZ"
390 GOTO 50
```

Lines 120 to 180 set the data extraction parameters and the specify what index information to gather.

Line 220 recalls the existing ZPL label and the subsequent lines insert the extracted index data into numbered fields.

Example 5

This example illustrates how to communicate with different ports using different channels – here the serial port is used rather than the console. Take notice of how the INPUT and PRINT commands are used.

```
10 CLOSE #0
20 OPEN #1 : NAME "SER"
30 OPEN #2 : NAME "ZPL"
40 INPUT #1 : A$
50 PRINT #2 : "^XA^FO20,20^A0N,50,50^FD"&A$&"^FS^XZ"
```

Line 10 of this sample closes the console (the console is the default channel when the interpreter is started). This should be done when there is a controlling character being transmitted with the string – for example, the Ctrl-C keystroke combination will terminate a program if it is received from the console.

Line 20 opens channel 1 as the serial port.

Line 30 opens channel 2 as the ZPL engine.

Line 40 indicates to the program which channel is going to be receiving the data. In this example, it is the serial port on channel one. If no channel value is given, the program will default to the console.

Line 50 prints the data that you enter to the label engine, specified on channel 2.

SECTION TWO
ZBI Programming Commands

Using Section Two: ZBI Programming Command Reference

This section contains a complete alphabetical listing of the ZBI commands supported by the X.10 firmware release. If you are a user of a previous version of the Zebra Printer firmware, ZBI code *will not be recognized*.

The text in this section is arranged under the following headings:

Description: Under this heading you will find a description of how the command is used, what it is capable of, and any defining characteristics it has.

Format: The *format* is how the command is arranged and what parameters it contains. For example, the AUTONUM command starts the auto-numbering option. The format for the command is AUTONUM <A>,<B>. The <A> and <B> are *parameters* of this command and are replaced with values determined by the user.

Parameters: If a command has values that the user can define to make command function more specific, they are listed under this heading. Still using the AUTONUM example, the <A> parameter is defined as:

```
<A> = number used to start the auto-numbering sequence
```

Example: When a command is best clarified in context, an example of the ZBI code is provided. Text indicating parameters, exact code to be entered, or data returned from the host is printed in the Courier font to be easily recognizable. An example of AUTONUM code is:

```
>AUTONUM 10,5
10 PRINT "HELLO WORLD"
15 GOTO 10
```

In an example, the > symbol indicates a line of code the user will enter.

Comments: This section is reserved for notes that are of value to a programmer, warnings of potential command interactions, or command-specific information that should be taken into consideration. An example comment could be: *This is a program command and must be preceded by a line number*.

A complete listing of ZBI commands for X.10 firmware begins on the next page.

AND

Description: The AND statement is a Boolean operator. If both of the expressions are true, the result is true; otherwise the result is false.

Format:

```
<Boolean expression> AND <Boolean expression>
```

Comments: This is a Boolean operator that is used in conjunction with Boolean expressions.

Arrays

Description: An array is a collection of values used by a program. Arrays share the following characteristics:

- arrays are allowed for both integer and string variables.
- indexes of arrays are accessed through parentheses.
- array indexes start at 1 and end at the length of an array (e.g. variable 3 will return the value in the third location of the variable array).
- one- and two-dimensional arrays are allowed. Two-dimensional arrays are referenced with two indexes in parenthesis, separated by a comma.
- arrays can only be re-dimensioned by a call to DECLARE, which will destroy the original array.
- array size is only limited by the size of the memory heap allocated.
- if an array cannot be allocated, an error message will be displayed – *Error: Heap overflow*.
- if the user attempts to access an array outside of its limits, an error message will be displayed – *Error: Invalid array access*.

AUTONUM

Description: This command automatically generates sequential program line numbers. This feature is disabled by overwriting the current line number and entering the desired interactive mode commands, or leaving the line blank.

Format:

```
AUTONUM A,B
```

Parameters:

A = the number used to start the auto-numbering sequence

B = the automatic increment between the new line numbers

Example:

```
>AUTONUM 10,5
10 PRINT "HELLO WORLD"
15 GOTO 10
```

Comments: The two lines are automatically started with the AUTONUM parameters; in this case, the first line is started with 10 and each subsequent line in incremented by 5.

This is an interactive command that takes effect as soon as it is received by the printer.

Boolean Expression

Description: A Boolean expression holds 0 (zero) as false and non-zero as true.

Formats:

```
<string expression> <Boolean compare> <string expression>

<# expression> <Boolean compare> <# expression>

(<Boolean expression>)
```

Comments: # indicates a numeric expression. A numeric expression cannot be compared to a string expression. If attempted, an error message will be displayed – *Error: Poorly formed expression*.

A numeric expression can be substituted for a Boolean expression where a value of 0 (zero) represents false and a non-zero value represents true.

Order of precedence for <, <=, >, >=, = are all the same order followed by NOT, AND, and OR, in that order. Items with the same order of precedence will be processed from left to right.

BREAK

Description: This command is available only when the DEBUG function has been activated. When DEBUG is on, BREAK will halt processing. RUN will start the program from the beginning. RESTART will allow the program to continue from where it left off.

Format:

```
BREAK
```

Comments: This is a program command that is preceded by a line number.

Channels

Description: I/O commands can be issued to channels to direct the commands to specific ports. The data-input port is specified through the ZPL commands that start the ZBI interpreter. It is through this port that all the interactive communications take place.

Format:

```
#<channel expression>
```

Parameters:

```
#<channel expression>
```

Accepted Values: 0 to 9
Default Value: 0

CLOSE

Description: This command is implemented to close specific ports that are in use. If a port is open on a channel and the CLOSE command is entered, the port will close and return to communicating with the ZPL buffer.

Format:

```
CLOSE <channel expression>
```

Parameters:

<channel expression>
Accepted Values: 0 through 9
Default Value: will not be used

Example:

```
>CLOSE #1
```

Comments: This can be an interactive command that takes effect as soon as it is received by the printer, or a program command that is preceded by a line number.

CLRERR

Description: This command sends a message to clear the error flag to the printer. The error flag will be reset if the error is non-permanent.

Format:

```
10 CLRERR
```

Comments: This is a program command that is preceded by a line number.

CTRL-C

Description: Sending 03 to port 0 or using the Ctrl-C keystroke combination will terminate any ZBI program currently running.

DEBUG

Description: When DEBUG is on, the TRACE and BREAK options are enabled. When DEBUG is off, the TRACE and BREAK options are disabled.

Format:

```
DEBUG <ON/OFF>
```

Parameters:

<ON/OFF> = toggles the debug mode *on* or *off*

Comments: This can be an interactive command that takes effect as soon as it is received by the printer, or a program command that is preceded by a line number.

Declaration

Description: Some common characteristics of declarations are:

- The names of variables are typically established at the beginning of a ZBI program. When declared, integer variables will be initialized to 0. When declared, string variables will be initialized to empty strings (“ ”).
- An explicit declaration deletes and reassigns any previously declared variable.
- An implicit declaration uses a variable name that is not previously defined. The value of this variable will be undefined.
- Non-array string and integer variables may be declared implicitly or explicitly.
- Arrays must be explicitly defined.

DECLARE

Description: This is the explicit method of declaring a variable and is typically performed at the beginning of a program. Arrays are specified by placing a set of closed parentheses around a numeric expression of the array size to be allocated.

Format:

```
DECLARE <type> <variable name> [,<variable name>]*
```

Parameters:

<type> = numeric or string

<variable name> = the variable being declared, which may be an array

Example:

```
10 DECLARE NUMERIC A ! Declare an integer
20 DECLARE STRING A$ ! Declare a string
30 DECLARE NUMERIC My_Array(10) ! Declare an
   integer array of size 10
40 DECLARE STRING My_Str_Array$(10) ! Declare a
   string array with 10 strings
50 DECLARE NUMERIC A, B(10), C ! Declare multiple
   integer variables
60 DECLARE STRING A$, B$, C$ ! Declare multiple
   string variables
70 DECLARE STRING A2D$(5,5) ! Declare A2-D array
```

Comments: This can be an interactive command that takes effect as soon as it is received by the printer, or a program command that is preceded by a line number.

DELETE

Description: This command removes a specified file from the printer’s memory.

Format:

```
DELETE <"filename">
```

Parameters:

<"filename"> = the name of the file to be deleted. Drive location and filename must be in quotation marks.

Example:

```
>DELETE "E:PROGRAM1.BAS"
```

Comments: This is an interactive command that takes effect as soon as it is received by the printer.

DIR

Description: This command, with no filter included, prompts the printer to list all the ZBI programs residing in all memory locations in the printer.

Including a filter signals the printer to limit the search; including a drive location signals the printer to search in only one location.

Asterisks (*) are used as wild cards. A wild card (*) will find every incidence of a particular request. The example here, `DIR "B:*.BAS"` signals the printer to search for every file with a `.BAS` extension in B: memory.

Format:

```
DIR ["filter"]
```

Parameters:

`["filter"]` = the name of the file to be accessed (optional). Drive location and filename must be in quotation marks.

Examples:

```
>DIR
>DIR "E:BASIC1.BAS"
>DIR "B:*.BAS"
>DIR "R:BASIC3.BAS"
```

Comments: This is an interactive command that takes effect as soon as it is received by the printer.

DO-LOOP

Description: Processing of the loop is controlled by a `<WHILE/UNTIL>` expression located on the `DO` or `LOOP` line.

Processing a `WHILE` statement is the same on either the `DO` or `LOOP` lines. The Boolean expression is evaluated and if the statement is true, the `LOOP` will continue at the line after the `DO` statement. Otherwise, the line after the corresponding `LOOP` will be the next in line to be processed.

Processing an `UNTIL` statement is the same on either the `DO` or `LOOP` lines. The Boolean expression is evaluated and if the statement is false, the `LOOP` will continue at the line after the `DO` statement. Otherwise, the line after the corresponding `LOOP` will be the next to be processed.

If `<WHILE/UNTIL>` is on the `LOOP` line, the `BODY` of the loop will be executed before the Boolean expression is evaluated.

If neither the `DO` or `LOOP` line has a `<WHILE/UNTIL>` statement, the loop will continue indefinitely.

`DO-LOOP`s may be nested but cannot overlap. The `DO-LOOP` command has two formats.

Format 1:

```
DO <WHILE/UNTIL> <Boolean expression>
~~BODY~~
LOOP
```

Example of Format 1:

```
>10 DO WHILE A$="70"
>20 INPUT A$
>30 LOOP
```

Format 2:

```
DO
~~BODY~~
LOOP <WHILE/UNTIL> <Boolean expression>
```

Example of Format 2:

```
>10 DO
>20 INPUT A$
>30 LOOP UNTIL A$="EXIT"
```

Comments: This is a program command that is preceded by a line number.

ECHO

Description: When the console mode is enabled, this command controls whether the printer will echo the characters back to the communications port. If ECHO ON is entered, keystroke results are returned to the screen. If ECHO OFF is entered, keystroke results are not returned to the screen.

Format:

```
ECHO <ON/OFF>
```

Parameters:

<ON/OFF> = toggles the ECHO command on or off

Comments: This can be an interactive command that takes effect as soon as it is received by the printer, or a program command that is preceded by a line number.

END

Description: The END command terminates any program currently running. When the END command is received, the interpreter will return to interactive mode.

Format:

```
END
```

Example:

```
>10 PRINT "THIS PROGRAM WILL TERMINATE"
>20 PRINT "WHEN THE END COMMAND IS RECEIVED"
>30 END
```

Comments: This is a program command and will be preceded by a line number.

! (EXCLAMATION MARK)

Description: The exclamation mark is the marker for adding comments to the end of numbered programming lines. Any text following the ! will be ignored when the line or command is processed.

Format:

```
![comment text]
```

Example:

```
10 LET A=10 ! Indicates number of labels to print
```

EXIT

Description: This command is used to exit the DO and FOR loops.

Format:

```
EXIT <DO/FOR>
```

Comments: This is a program command that is preceded by a line number.

FOR-LOOP

Description: The FOR loop will assign the numeric variable to the value of the first expression. The NEXT line will increment the numeric variable by the STEP amount.

If the numeric variable is less than the second expression after being incremented by the NEXT line, the program will resume running at the line following the FOR line.

When the STEP portion is omitted, the STEP value will be 1. However, if the second expression is greater than the first expression, it will default to -1.

Format:

```
FOR <# variable> = <# expression> TO <# expression>
   [STEP <# expression>]
~~BODY~~
NEXT <# variable>
```

Parameters:

<# variable> = indicates a numeric variable is used
<# expression> = indicates a numeric expression is used

Example:

```
10 FOR X=1 TO 10 STEP 1
20 PRINT X; ":ZBI IS FUN"
30 NEXT X
```

Comments: FOR-LOOPs may be nested but cannot overlap. Variables cannot be reused by the nested loops. This is a program command that is preceded by a line number.

Functions

Description: Functions built into this interpreter can be used in expressions only. The function names are not case sensitive.

If input parameters exist, they are enclosed in parentheses. If no parameters exist, no parentheses are used.

Variables referenced in the functions may be substituted by functions or expressions of the same type. If the function name ends with a $, it will return a string value. Otherwise it will return a numeric value.

Specifying the wrong type of parameter will return either *Syntax Error* or *Poorly formed expression error*.

Integer Functions

The integer functions listed below return a numeric value:

DATE

This function returns the current date in YYYYDDD, where YYYY is the year and DDD is the number of days since the beginning of the year. If the Real Time Clock is not installed, 0 will be returned. For this example, assume the current date is January 1, 2000:

```
>10 PRINT DATE
>RUN
```

The result is:

```
2000001
```

DATAREADY(A)

This function returns the numeral 1 if data is ready on port A, and returns a 0 if there is no data available. Example:

```
>10 PRINT DATAREADY(0)
>RUN
```

The result, assuming no data is waiting, is:

```
0
```

ISERROR

This function returns a non-zero value if there is an internal error set in the printer. Otherwise, the numeral returned will be 0. Example:

```
>10 PRINT ISERROR
>RUN
```

The result is:

```
0
```

ISWARNING

This function returns a non-zero value if there is an internal warning set in the printer. Otherwise, the numeral returned will be 0. Example:

```
>10 PRINT ISWARNING
>RUN
```

the result is:

```
0
```

LEN(A$)

This function returns the length of the string A$. Example:

```
>10 LET A$="Hello World"
>20 PRINT LEN(A$)
>RUN
```

The result is:

```
11
```

MAX(X,Y)

This function returns the maximum algebraic value of X or Y. If X is greater than Y, the value of X will be returned. Otherwise, the value of Y will be returned. Example:

```
>10 LET A=-2
>20 LET B=1
>30 PRINT MAX(A,B)
>RUN
```

The result is:

```
1
```

MAXLEN(V$)

This function returns the maximum length for the string V$, which is always 255. This value is independent of V$ but remains for compatibility with the ANSI specification. Example:

```
>10 LET A$="Hello"
>20 PRINT MAXLEN(A$)
>RUN
```

The result is:

```
255
```

MAXNUM

This function returns the largest number represented by this machine: 2,147,483,647. Example:

```
>10 PRINT MAXNUM
>RUN
```

The result is:

```
2147483647
```

MIN(X,Y)

This function returns the minimum algebraic value of X or Y. If X is less than Y, the value of X will be returned. Otherwise, the value of Y will be returned. Example:

```
>10 LET A=-2
>20 LET B=0
>30 PRINT MIN(A,B)
>RUN
```

The result returned is:

```
-2
```

MOD(X,Y)

This function returns X Modulo Y (same as the remainder of X/Y). Example:

```
>10 LET A=9
>20 LET B=2
>30 LET C=-2
>40 PRINT MOD(A,B)
>50 PRINT MOD(C,A)
>RUN
```

The result is:

```
1
-2
```

ORD(A$)

This function returns the ASCII value of the first character of string A$. Example:

```
>10 LET A$="ABC"
>20 PRINT ORD(A$)
>RUN
```

The result is:

```
65
```

POS(A$,B$)

This function returns the location of the first occurrence of B$ in A$. If there is no occurrence, the result will be 0. Example:

```
>10 LET A$="ABCDD"
>20 LET B$="D"
>30 PRINT POS(A$,B$)
>RUN
```

The result is:

```
4
```

POS(A$,B$,M)

This function returns the location of the first occurrence of B$ in A$ starting at the position of M. If there is no occurrence, the result is 0. Example:

```
>10 LET A$="Hello World"
>20 LET B$="o"
>30 PRINT POS(A$,B$,6)
>RUN
```

The result is:

```
8
```

TIME

This function returns the time past midnight (2400h) in seconds. If the Real Time Clock is not installed, 0 will be returned. Example:

```
>10 PRINT TIME
>RUN
```

The result, assuming the time is one minute past midnight, is:

```
60
```

VAL(A$)

This function returns the numeric value represented by A$. The conversion is done in the same fashion as the `INPUT` command. Example:

```
>10 LET A$="123"
>20 LET C=VAL(A$)
>30 PRINT C
>RUN
```

The result is:

```
123
```

String Functions

The string functions listed below return a string value:

CHR$(M)

This function returns the character at position M of the extended ASCII table. Using values of M greater than 255 will have its value modulated by 255. M may not be 0. The numeral 1 will be substituted. Example:

```
>10 LET A=97
>20 PRINT CHR$(A)
>30 PRINT CHR$(353) !Note that 353 is modulated to 97
>RUN
```

The result is:

```
a
a
```

DATE$

This function returns the current date in string form YYYYMMDD. If the Real Time Clock is not installed, no data will be returned. Example:

```
>10 PRINT DATE$
>RUN
```

The result, assuming the date is January 1, 2000 is:

```
20000101
```

EXTRACT$(A$,B$,C$)

This function returns the string contained in A$ between the first occurrence of B$ and the following occurrence of C$. If B$ or C$ do not exist in A$, a NULL string will be returned. Example:

```
>10 LET A$="HELLO"
>20 LET B$="L"
>30 LET C$="O"
>40 PRINT EXTRACT$(A$,B$,C$)
```

The result is:

```
L
```

Example 2:

```
>10 LET A$="HELLO"
>20 LET B$="H"
>30 LET C$=""
>40 PRINT EXTRACT$(A$,B$,C$)
>RUN
```

The result of Example 2 is:

```
ELLO
```

Example 3:

```
>10 LET A$="Hello"
>20 LET B$="C"
>30 LET C$="F"
>40 PRINT EXTRACT$(A$,B$,C$)
>RUN
```

In Example 3, nothing is returned.

LCASE$(A$)

This function returns a string corresponding to A$ in lowercase letters. Non-character values will not be changed. Example:

```
>10 LET A$="Zebra Technologies"
>20 PRINT LCASE$(A$)
>RUN
```

The result is:

```
zebra technologies
```

LTRIM$(A$)

This function returns the string A$ with all leading spaces removed. Example:

```
>10 LET A$=" Hello"
>20 PRINT LTRIM$(A$)
>RUN
```

The result is:

```
Hello
```

REPEAT$(A$,M)

This function returns a string containing M copies of A$. Example:

```
>10 LET X=3
>20 LET A$="Hello"
>30 PRINT REPEAT$(A$,X)
>RUN
```

The result is:

```
HelloHelloHello
```

RTRIM$(A$)

This function returns a string created from A$ with trailing spaces removed. Example:

```
>10 LET A$="Hello "
>20 LET B$="World"
>30 PRINT RTRIM$(A$);
>40 PRINT B$
>RUN
```

The result is:

```
HelloWorld
```

STR$(X)

This function converts numeric type X to a string. Example:

```
>10 LET A=53
>20 PRINT STR$(A)
>RUN
```

The result is:

```
53
```

TIME$

This function returns the time of day in format HH:MM:SS. If the Real Time Clock is not installed, no data will be returned. Example:

```
>10 PRINT TIME$
>RUN
```

The result, assuming it is 10 a.m., is:

```
10:00:00
```

UCASE$(A$)

This function returns a string created from A$ with all characters in uppercase. Non-character values will not be changed. Example:

```
>10 LET A$="Zebra Technologies"
>20 PRINT UCASE$(A$)
>RUN
```

The result is:

```
ZEBRA TECHNOLOGIES
```

GOTO

Description: The GOTO statement is used to direct the interpreter to a specific line number. GOTO is followed by a line number that the program will attempt to process next. Upon executing the GOTO statement, the interpreter will continue running at the line number specified following GOTO. If the line number referenced does not exist, an error message will be displayed – *Error: Line does not exist*.

Format:

```
GOTO
```

Example:

```
>10 PRINT "Zebra Printers"
>20 GOTO 10
```

Comments: The result will display “Zebra Printers” until the program is halted. This is a program command and must be preceded by a line number.

GOSUB-RETURN

Description: GOSUB is followed by a line number that the program will attempt to process next. Upon executing the GOSUB statement, the interpreter will continue running at the line number specified following GOSUB. If the line number referenced does not exist, an error message will be displayed – *Error: Line does not exist*.

Before executing the next line, the GOSUB command will store the line number of the GOSUB line. When the RETURN statement is called, the program will move back to the next line following the GOSUB.

Executing a RETURN statement without a corresponding GOSUB statement will cause an error message to be displayed – *Error: Invalid RETURN statement*.

GOSUB statements can be nested.

Format:

```
GOSUB
RETURN
```

Example:

```
>10 PRINT "Call Subroutine"
>20 GOSUB 1000
>30 PRINT "Returned from Subroutine"
>40 END
>1000 PRINT "In Subroutine"
>1010 RETURN
```

Comments: These are program commands and must be preceded by line numbers.

IF Statements

Description: If the value of the <Boolean expression> in an IF statement is true and a program line follows the keyword THEN, this program line will be executed. If the value of the Boolean expression is false and a program line follows the keyword ELSE, this program line will be executed. If ELSE is not present, then execution will be continued in sequence, with the line following the END IF statement.

Nesting of blocks is permitted, subject to the same nesting constraints as DO-LOOPs (no overlapping blocks).

ELSE IF statements are treated as an ELSE line followed by an IF line, with the exception that the IF shares the END IF line of the original IF statement.

Format:

```
IF <Boolean expression> THEN
~~BODY~~
[ELSE IF <Boolean expression> THEN
~~BODY~~]*
[ELSE
~~BODY~~]
END IF
```

Example:

```
>10 IF A$="0" THEN
>20 PRINT "ZBI IS FUN"
>30 ELSE IF A$="1" THEN
>40 PRINT "ZBI IS EASY"
>50 ELSE
>60 PRINT "X=0"
>70 END IF
```

Comment: This is a program command that is preceded by a line number.

INBYTE

Description: This command forces the interpreter to pause until data is available. Use the DATAREADY function to determine if there is data on the port.

Format:

```
INBYTE [channel expression:] <A>
```

Parameter:

<A> = numeric or string expression. The received value will replace the current value held in the variable.

Example:

```
>10 INBYTE A$ !takes one byte (char) from port #1
>20 PRINT A$ !prints the character to the console
```

Comments: In this example, the interpreter will pause until the data is entered and then continue processing. This command will input all bytes in a string or integer, including control codes.

This can be an interactive command that takes effect as soon as it is received by the printer, or a program command that is preceded by a line number.

INPUT

Description: If the variable is numeric and the value entered cannot be converted to a number, it will be written as 0. This operation scans the data from left to right, shifting any number into the variable. It ignores any other character except the return character, which terminates the input, or Ctrl-C (^C) which terminates the program. The variable can be in string or numeric form.

Format:

```
INPUT [<channel expression>:] <variable> [,variable]*
```

If the `[<channel expression>:]` is omitted, the default port is 0. If an invalid port is specified, an error condition is returned in the form of *Error: Invalid port*.

Example 1:

```
>10 INPUT A
>20 PRINT A
```

If the user entered 1234567891011, the number sent to display will be modulated by the `MAXNUM` value.

Example 2:

```
>10 OPEN #1: NAME "ZPL"
>20 PRINT #1: "~HS"
>30 FOR I = 1 TO 3
>40 INPUT #1: A$
>50 PRINT A$
>60 NEXT I
```

In Example 2, a host status will be printed to the console after submitting the host status request ~HS to the ZPL port.

The Input/Output command of the ZBI interpreter will be limited to the communications ports. File I/O is not supported.

Comments: This can be an interactive command that takes effect as soon as it is received by the printer, or a program command that is preceded by a line number.

LET

Description: The `LET` command is used to assign value to a specific variable. The expression is evaluated and assigned to each variable in the variable list.

Format:

```
LET <variable> [,<variable>]* = <expression>
```

The variable types must match the expression type or an error message will be displayed – *Error: Variable types must be the same.*

When a value is assigned to a string variable with a sub-string qualifier, it replaces the value of the sub-string qualifier. The length of the value of the string variable may change as a result of this replacement.

Examples:

```
>10 LET A$= "1234"
>15 LET A$(2:3)= "55" ! A$ NOW = 1554
>20 LET A$(2:3)= "" ! A$ NOW = 14

>10 LET A$= "1234"
>15 LET A$(2:3)= A$(1:2) ! A$ NOW = 1124

>10 LET A$= "1234"
>20 LET A$(2:1)= "5" ! A$ NOW = 15234
```

Comments: This can be an interactive command that takes effect as soon as it is received by the printer, or a program command that is preceded by a line number.

LIST

Description: This command numerically lists the program lines currently in memory.

Format:

```
LIST [RANGE]
```

Parameters:

`[RANGE]` = an optional request for specific lines (or a line) of code. `[RANGE]` is a single pair of numbers separated by a hyphen.

Examples:

```
>LIST 20
>LIST 90-135
```

The printer returns line 20, or in the second example, lines 90 through 135.

Comments: This is an interactive command that takes effect as soon as it is received by the printer.

LOAD

Description: This command transfers a program file previously stored in the printer's memory and opens it in the ZBI Program Memory.

If the program file does not exist, the ZBI Program Memory is cleared and no program is opened. An error message will be displayed – *Error: Invalid file name*.

Format:

```
LOAD <"filename">
```

Parameter:

`<"filename">` = the name of the file to be loaded into memory. Drive location and filename must be in quotation marks.

Example:

```
>LOAD "PROGRAM1.BAS"
>LOAD "E:PROGRAM1.BAS"
```

Comments: This is an interactive command that takes effect as soon as it is received by the printer.

NEW

Description: This command clears the interpreter's memory, including the line buffer and variables, but not any open ports.

Format:

```
NEW
```

Example:

```
>NEW
```

Comments: This is an interactive command that takes effect as soon as it is received by the printer.

NOT

Description: The `NOT` statement is a Boolean operator. If the Boolean expression is true, the result is false. If the Boolean expression is false, the result is true.

Format:

```
NOT <Boolean expression>
```

Example:

```
>10 LET A=1
>20 IF NOT A=0 THEN
>30 PRINT A
>40 END IF
>RUN
```

Comments: This is a Boolean operator that is used in conjunction with Boolean expressions.

Numeric Expressions

Description: Base numerical expression can be either a constant, variable, or another numerical expression closed in by parentheses. Overflow cannot be detected. The result will be undefined. The five types used (addition, subtraction, multiplication, division, and exponentiation) are listed below.

1. + (addition) Expressions involving addition use the following format:

```
<numerical expression>+<numerical expression>
```

2. - (subtraction) Subtraction expressions use the following format:

```
<numerical expression>-<numerical expression>
```

3. * (multiplication) Multiplication expressions use the following format:

```
<numerical expression>*<numerical expression>
```

4. / (division) Division expressions use the following format:

```
<numerical expression>/<numerical expression>
```

5. ^ (exponentiation) Exponentiation expressions use the following format:

```
<numerical expression>^<numerical expression>
```

In mathematics, order of precedence describes in what sequence items in an expression will be processed. All expressions have a predefined order of precedence.

The order of precedence is ^, * /, + -.

* and / have the same precedence, and + and - have the same precedence. Items with the same order of precedence will be processed from left to right.

For example, the following expression 5+(8+2)/5 would be processed as 8+2=10, followed by 10/5=2, then 5+2 to give a result of 7.

Functions and parenthesis always have the highest order of precedence, meaning that they will get processed first. The remaining items are specific to the type of expression it is (Boolean, numeric, and string).

Maximum value: 2,147,483,647

Minimum value: -2,147,483,648

Comments:

- No floating point is supported.
- Variable names must start with a letter and can include any sequence of letters, digits, and underscore.
- Function and command names may not be used as variable names.
- Variable names are not case sensitive and will be converted to uppercase by the interpreter.
- When using division, the number will always be rounded down. For example, 5/2=2.

ON ERROR

Description: The `ON ERROR` command can be used to prevent a program from halting in the event of an error. If an error occurs in a previous line during program execution, the `ON ERROR` statement calls the `GOTO` or `GOSUB` statement and allows the program to continue.

Format:

```
ON ERROR <GOTO/GOSUB> LN
```

If there is no error, this line will be ignored.

Example:

```
30 LET A = B/C
40 ON ERROR GOTO 100
...
100 PRINT "DIVIDE BY ZERO OCCURRED"
110 LET A = 0
120 GOTO 50
...
```

Comments: This is a program command that is preceded by a line number.

OPEN

Description: This command is used to open a port for transmitting and receiving data.

Format:

```
OPEN #<channel expression>: NAME <string expression> [, ACCESS <ACCESS type>]
```

Parameters:

`<channel expression>` =
Accepted Values: 0 to 9
Default Value: a port must be specified

`<string expression>` = port name to open (SER, PAR, or ZPL)

`<ACCESS type>` =
INPUT for receiving only
OUTPUT for transmitting only
OUTIN for transmitting and receiving

A channel must be specified; the default value cannot be used. Access type OUTIN is used if access type is not specified.

Example:

```
>10 OPEN #1 : NAME"ZPL"
```

If there are conflicts opening the port, an error message will be displayed – *Error: Unable to open port*. If the port is already open, an error message will be displayed – *Error: Port already opened*.

The port being opened will no longer allow data to pass directly into its buffer, it will be disconnected, and the interpreter will now control the data flow.

Data already in the buffer will stay in the buffer.

Comments: This can be an interactive command that takes effect as soon as it is received by the printer, or a program command that is preceded by a line number.

OR

Description: The OR statement is a Boolean operator. If either of the expressions are true, the result is true; otherwise the result is false.

Format:

```
<Boolean expression> OR <Boolean expression>
```

Comments: This is a Boolean operator that is used in conjunction with Boolean expressions.

OUTBYTE

Description: This command will output all bytes in a string, including control codes.

Format:

```
OUTBYTE [<channel expression>:] <A>
```

Parameters:

<A> = a numeric or string expression.

If the parameter is a numeric expression, it must be a value of 0 through 255. If not, it will be truncated. For a string, the first character will be used. In case of a NULL string, 0 will be sent.

Example:

```
>Let A$="Hello"
>OUTBYTE A$
```

Result:

```
H
```

Comments: This can be an interactive command that takes effect as soon as it is received by the printer, or a program command that is preceded by a line number.

Ports <CHANNEL EXPRESSION>

Description: Each printer has a list of ports that may be opened by the interpreter.

Each port will be given a three-letter abbreviation that will be used with the OPEN command to open that port. The standard port names are “SER” for the serial port and “PAR” for the parallel port. The port reference “ZPL” opens a channel to the printer’s formatting engine that functions as a port not controlled by the ZBI interpreter.

PRINT

Description: This command sends data to the printer to be printed.

Format:

```
PRINT [channel expression:] <expression> [,or; <expression>]* [;]
```

The expression can be either a string or a numeric expression.

Using a , to separate expressions will add a space between them.

Using a ; to separate expressions will not put a space between them.

Using a ; at the end of a line will end the print statement without a new line.

Example:

```
>10 LET A$ = "This is an example"
>20 LET B$ = "of the PRINT Command."
>30 PRINT A$, B$ ! adds a space between expressions
>40 PRINT A$; B$ ! no space added
>RUN
```

Result:

```
This is an example of the PRINT Command.
This is an exampleof the PRINT Command.
```

Comments: This can be an interactive command that takes effect as soon as it is received by the printer, or a program command that is preceded by a line number.

REM

Description: A numbered “remark” line is started with REM and can include any text in any form after it. This line will be ignored by the interpreter.

Format:

```
REM [comment]
```

Example:

```
10 REM COMMAND LINES 20-100 PRINT A LABEL
```

Comments: Remarks are used for program description and can be included as a separate program line or appended to the end of a program line. Also used for internal comments is the exclamation mark (!) statement.

RENUM

Description: This command renumbers the lines of the program being edited. RENUM can reorganize code when line numbers become over- or under-spaced. The line references following GOTO and GOSUB statements are renumbered if they are constant numeric values. Renumbering does not occur if the line number are outside of the range limits of 1 to 10000.

Format:

```
RENUM <A>,<B>
```

Parameters:

<A> = the number to start the renumbering sequence

<B> = the automatic increment between the new line numbers

Example:

```
13 LET A=6
15 LET B=10
17 GOTO 13

>RENUM 10,5

10 LET A=6
15 LET B=10
20 GOTO 10
```

Comments: This is an interactive command that takes effect as soon as it is received by the printer.

RESTART

Description: If a program was halted by Ctrl-C or a BREAK command, the RESTART command can be used to reactivate the program from the point it stopped. RESTART functions similar to RUN, except the program will attempt to restart from the point it was last terminated. It will also work in conjunction with the STEP command, picking up where STEP ends.

Format:

```
RESTART
```

Comments: If the program has not been run or is finished, RESTART will run the program from the beginning. This is an interactive command that takes effect as soon as it is received by the printer.

RUN

Description: This command allows the program being edited to be activated, starting with the lowest line number. If a line does not specify a new line to go to, the next line in numeric order is processed. When a higher line number does not exist, the RUN command stops.

Format:

```
RUN
```

Example:

```
>10 PRINT "ZBI"
>15 PRINT "Programming"
>RUN
```

Result:

```
ZBI
Programming
```

Comments: Ports that are open when the application is activated will remain open after the application has terminated. Variables will also remain after the application has terminated.

This is an interactive command that takes effect as soon as it is received by the printer.

SEARCHTO$ (A,B$)

Description: This function performs a search up to a string, which will be defined by B$ on port A. The string the search yields is displayed.

Format:

```
SEARCHTO$(A,B$)
```

Parameters:

`A` = port number (0 to 9) that requested data is sent to.

`B$` = string variable or string array. If `B$` is an array, this command will search for all non-null strings in the `B$` array.

Example:

```
10 LET A$=SEARCHTO$(0,"Hello")
```

`There are several ways to display Hello World` is the text string being searched in this example. `A$` will then equal `Hello`, perform the search, and only `World` will remain on the port.

SEARCHTO$ (A,B$,C)

Description: This function works similar to `SEARCHTO$(A,B$)` and performs a search up to a string, which will be defined by `B$` on port A. The string the search yields is displayed. Unused characters up to the search string are sent to port C.

Format:

```
SEARCHTO$(A,B$,C)
```

Parameters:

`A` = port number (0 to 9) the requested string is sent to.

`B$` = string variable or string array. If `B$` is an array, this command will search for all non-null strings in the `B$` array.

`C` = port number (0 to 9) unused characters are sent to.

Example:

```
10 LET A$=SEARCHTO$(0,"Hello",1)
```

`There are several ways to display Hello World` is the text string being searched for this example. `A$` will then equal `Hello`, perform the search, and only `World` will remain on the port. The unused characters `There are several ways to display` are sent to port 1.

SETERR

Description: This command sends a message to the printer to set the error flag. A logical interpreter flag will be triggered in the printer. This error will be referenced as a BASIC Forced Error.

Format:

```
SETERR
```

Comments: This is a program command and must be preceded by a line number.

SLEEP

Description: This command specifies the time that the interpreter pauses for label generation to receive greater priority. This command may be sent to the printer after sending a label format to be printed. The interpreter pauses in its processing for the amount of time specified.

Format:

```
SLEEP <A>
```

Parameters:

<A> = the time in seconds (0 to 500) the interpreter will pause.

Example:

```
>10 SLEEP 450
```

Comments: This is a program command and must be preceded by a line number.

STEP

Description: If a program was stopped by a BREAK command, STEP will attempt to execute the program one line from where it last ended. If the program has not been run or has been completed, this will execute the lowest numbered line.

Format:

```
STEP
```

Example:

```
>10 PRINT "Hello World"
>20 BREAK
>30 PRINT "30"
```

Entering STEP would cause line 10 to be executed.

Entering RUN followed by STEP would cause:

RUN – Execute to line 20 and stop.

STEP – Execute line 30 and stop.

Comments: This is an interactive command that takes effect as soon as it is received by the printer.

STORE

Description: This command saves the program currently in memory as the specified filename. The following format is used:

```
STORE <"filename">
```

Parameters:

<"filename"> = the name of the file to be stored. Drive location and filename must be in quotation marks.

Example:

```
>STORE "E:ZEBRA1.BAS"
```

Comments: For a filename to be valid, it must conform to the 8.3 Rule: each file must have a no more than 8 characters in the filename and have a 3-character extension. Here the extension will always be .BAS (e.g. MAXIMUM8.BAS).

This is an interactive command that takes effect as soon as it is received by the printer.

STRING CONCATENATION (&)

Description: The basic string expression may be either a constant or a variable, and the concatenation (&) is supported.

Using the concatenation operator will add the second string to the first string.

Format:

```
<string expression> & <string expression>
```

Example:

```
10 LET A$= "ZBI "
20 LET B$= "Programming"
30 LET C$= A$ & B$
40 PRINT C$
```

Result:

```
ZBI Programming
```

Comments: If the concatenation would cause the string to be greater than the maximum possible length of a string (255 characters), the first string is returned and an error message will be displayed – *Error: String size limit exceeded*.

STRING VARIABLE

Description: Maximum length: 255 characters

Variable names must start with a letter and can include any sequence of letters, digits, and underscore. The variable ends with a $.

Function and command names may not be used as a variable name.

Variable names are not case sensitive and will be converted to uppercase by the interpreter.

SUB-STRINGS

Description: Using a sub-string operator on a string allows a specific portion of the string to be accessed. To determine the coordinates of the string portion to be used, count the spacing from the beginning to the end of the string, including spaces.

Format:

```
StringVariable$(A:B)
```

Parameters:

A = the position of the first character in the desired string.
B = the position of the last character in the desired string.

Example:

```
>10 LET A$="Zebra Quality Printers"
>20 LET B$=A$(1:13)
>30 PRINT B$
```

Result:

```
Zebra Quality
```

To calculate the position of Zebra Quality Printers in line 10 above, “Z” occupies position 1, “e” occupies position 2, “b” occupies position 3, “r” occupies 4, “a” occupies 5, the space occupies 6, and so on.

Comments: If the A parameter is less than 1, it will be automatically assigned a value of 1. Since the string is calculated starting with 1, the A parameter cannot be less than 1.

If B is greater than the length of the string, it will be replaced with the length of the string. In this example, the B parameter could be no greater than 22.

If A is greater than B, a NULL string (“ ”), which points to the location of the smaller of A or the end of the string, will be returned. This is used when adding a string in the middle of another string without removing a character. Refer to the LET command.

TRACE

Description: This command is only valid when the DEBUG function is active.

Format:

```
TRACE <ON/OFF>
```

Parameters:

<ON/OFF> = controls whether TRACE is active (on) or disabled (off).

If DEBUG is activated and the TRACE command is on, trace details will be displayed. When any variables are changed, the new value will be displayed as follows:

```
Variable$=New Value
```

Every line processed will have its line number printed.

If DEBUG is *on*, the line number prints before the command line is executed and variables print as their values are assigned.

```
10 LET A=5
20 GOTO 40
30 PRINT "Error"
40 PRINT A
RUN
```

The output is:

```
<TRACE> 10
<TRACE> A=5
<TRACE> 20
<TRACE> 40
5
```

Comments: This can be an interactive command that takes effect as soon as it is received by the printer, or a program command that is preceded by a line number.

Zebra Technologies Corporation
333 Corporate Woods Parkway
Vernon Hills, Illinois 60061.3109 USA
Telephone +1 847.634.6700
Facsimile +1 847.913.8766

Zebra Technologies Europe Limited
Zebra House
The Valley Centre, Gordon Road
High Wycombe
Buckinghamshire HP13 6EQ, UK
Telephone +44 (0)1494 472872
Facsimile +44 (0)1494 450103

Customer order #45543L
Manufacturer part #45543LB