Reference Manual
Contents

Chapter 1 Introduction

Manual Conventions

Syntax Conventions

Printing Conventions

Chapter 2 Definitions

A B C D E F G H I J K L M

N O P Q R S T U V W X Y Z

Chapter 3 Statement Summary

Chapter 4 Keyword Dictionary

A B C D E F G H I J K L M

N O P Q R S T U V W X Y Z

Appendix A Error Codes

1-25 26-49 50-99 100-149

150-299 300-459 460-699 700-899

900-999 2000-2099

Appendix B ASCII Code Chart

Other Manuals

Installing and Using Manual

User's Guide

Basic Plus

{button www.htbasic.com,Inet("www.htbasic.com")}

Distributed with release 8.3

Definitions

A

Angle

Array

Array Name

ASCII file type

Definitions B

BDAT file type

Boolean Expression

Definitions

C

- COM Block
- COM Block Name
- COMPLEX
- Context

Definitions
D

Device Selector

DOS file type

Definitions

E

Event

Definitions
F

File Specifier

Full Array Specifier

Function Name

Definitions I

I/O PATH

Integer

Integer Array

Interface Select Code

Definitions L

Line Label

Line Number

Local Variable

Long

Definitions
M

Matrix

Definitions N

Numeric Array

Numeric Array Element

Numeric Constant

Numeric Expression

Numeric Name

Definitions O

Ordinary file

Definitions P

Path Specifier

Pen Number

Pipe Specifier

Precedence

Primary Address

Priority

PROG file type

Definitions
R

Real

Record Number

Definitions
S

Scientific Notation

Signal Number

Softkey Macro

Static

String

String Array

String Array Element

String Expression

String Literal

String Name

Sub-string

Subprogram Name

Subscript

Definitions

U

UNIX file type

Definitions
V

Variable Name

Vector

Volume Label

Volume Specifier

Chapter 1

Introduction

High Tech Basic (HTBasic) is a technical programming language compatible with Hewlett Packard's "Rocky Mountain" BASIC for HP 9000 Series 200/300 computers. It has extensive graphics, instrument control capabilities and interactive programming aids to speed program development. It is designed to offer powerful features and ease of use to engineers, scientists and other professionals having a range of programming experience from novice to expert.

The on line *Reference Manual* contains the following chapters:

- Chapter 1, Introduction
- Chapter 2, Definitions
- Chapter 3, Statement Summary
- Chapter 4, Keyword Dictionary
- Appendix A, Errors
- Appendix B, ASCII Code Chart

Chapter 1, "Introduction," is this chapter and introduces the manual layout. It contains conventions used throughout the manual and syntax rules used in Chapter 4.

Chapter 2, "Definitions," defines general terms and common syntactical units.

Chapter 3, "Statement Summary," lists all the HTBasic statements and indicates which can be executed from the keyboard, stored in a program, and included in an IF...THEN statement.

Chapter 4, "Keyword Dictionary," lists in dictionary fashion the HTBasic keywords. Entries include syntax diagram, sample statements, a description of the keyword's functionality and related keywords.

Appendix A, "Errors," lists each error number, cause, and in some cases, possible solutions.

Appendix B, "ASCII Code Chart," contains ASCII, decimal and hexadecimal values and IEEE-488 commands and addresses.

Manual Conventions

The following is an example "Keyword Dictionary" entry which explains the rules and conventions used throughout this manual.

KEYWORD

This line tells what the KEYWORD does.

Syntax: This line defines the syntax.

where: These lines, when present, further define parts of the syntax.

Sample: `These lines give samples using the KEYWORD.`

Description:

These paragraphs describe in greater detail how the **KEYWORD** is used. Several conventions are used to aid your understanding of the keyword. All terms used in the syntax definition are defined in one of two places. Commonly used terms, such as "numeric-expression," are defined at the beginning of the *Reference Manual*. Other terms are defined immediately after they are used, in the lines following the "where:".

See Also:

LISTS OTHER KEYWORDS RELATED TO THIS ONE.

Syntax Conventions

The key to understanding the syntax definitions is understanding the punctuation used in the definition. Braces and vertical bars are used to denote a list of choices. A construct like this:

{ ON | OFF }

means you must specify **ON** or **OFF** but not both. Do not enter the braces or the vertical bar. Square brackets are used to denote optional items. For example,

BEEP [frequency, duration]

means that BEEP may be entered alone or with the frequency and duration. Ellipses (three dots "...") are used to show that the preceding item can be optionally repeated any number of times. For example, in the definition

ALLOCATE item [,item...]

",item" can be optionally repeated one or more times. Single quotes, "'", are used around the square bracket symbols when they should be entered literally, instead of interpreted as optional item symbols. For example,

DIM string-name$ '['length']'

means that the bracket characters are part of the statement to dimension a string.

Words in lower-case, like "length" in the example above, are defined either later in the syntax definition itself or in the definitions at the start of the *Reference Manual*. Words in uppercase are keywords and should be entered exactly as shown. Keywords must be separated from one another by spaces. All other symbols should be entered exactly as shown. Spaces have been added in some definitions to improve readability.

Printing Conventions

Several printing conventions are used in this manual. In descriptions, keywords are shown in BOLD, UPPERCASE letters. (In other places, keywords are merely shown in uppercase.) Examples that show exactly what you should type or what the computer displays are shown in a fixed width font, such as

```
10 DIM A$[50]
```

Key names, IEEE-488 bus commands and operating system commands are shown in all uppercase, for example: ENTER.

Chapter 2
Definitions

This chapter contains definitions of 'Rocky Mountain BASIC' general terms and syntactical units.

Definition
Angle

Angles can be specified in radians or degrees. When specifying angles for graphic statements, the angle is relative to the positive x axis. Positive angles specify counter-clockwise movement about the origin.

Definition
Array

An array is a multi-dimensional ordered set of values. Each member of the set is called an array element. All the members of the set have the same simple data type which can be integer, long,   real, complex, or string. The dimension of the set is called the RANK of the array. Arrays may have a rank from one to six.

Local array variables are declared using INTEGER, LONG, REAL, STATIC, COMPLEX and DIM. ALLOCATE can be used to dynamically declare an array. COM can be used to declare a global array. Consult these entries in Chapter 4, the "Keyword Dictionary," to learn how to declare array variables. OPTION BASE is available to change the default lower bound for indices.

Definition
Array Name

The rules for naming an array are the same as for a variable (see Variable Name). Array variables and simple variables share the same name space. Thus, you cannot have a simple variable and an array variable with the same name in the same context.

Definition
ASCII file type

In the HTBasic manual set, the term ASCII file refers to a LIF ASCII file, not a DOS ASCII or UNIX ASCII ordinary file. A LIF ASCII file is a typed file which contains string items preceded by an item length, and followed by a pad byte when the string length is odd. Do not confuse the terms DOS ASCII, UNIX ASCII and LIF ASCII. A DOS ASCII file is an ordinary file which contains only printable characters and the end of each line is marked with a carriage return and line feed. A UNIX ASCII file is an ordinary file which contains only printable characters and the end of each line is marked with a line feed. HTBasic can read and write any of these file types. See CREATE and CREATE ASCII in Chapter 4, the "Keyword Dictionary."

Definition
BDAT file type

BDAT files are used to hold binary data and can be used to exchange data with HP BASIC. See CREATE BDAT in Chapter 4, the "Keyword Dictionary." Ordinary files can also be used to hold binary data.

Definition
Boolean Expression

A boolean expression is simply a numeric expression whose result is tested for zero/non-zero. If the result is zero, the expression is considered FALSE. If the result is non-zero, the expression is considered TRUE.

Definition
COM Block

A COM block is a set of one or more variables that may be shared (in "COMmon"), among one or more contexts. Each COM block is uniquely identified with a name (although one block is allowed to be nameless). COM block names are explained below.

The value of a COM variable is global in lifetime, however, the name of a COM variable is not global. To access COM variables, a context must include a COM statement which identifies the COM block and gives the names by which the variables will be known in that context. Thus, each context can give a different name to the same COM variable. COM variables are hidden from all contexts which do not include a COM statement accessing that COM block. See COM in Chapter 4, the "Keyword Dictionary."

Definition
COM Block Name

Rules for naming a COM block are the same as for a variable (see Variable Name).

Definition
COMPLEX

"Complex" is a data type. Other data types are integer, real, string, and I/O path. The Complex data type is a subset of all rational numbers. The particular subset depends on your computer. Most computers, including the IBM PC, use IEEE Std 754-1985 for Binary Floating point numbers. This gives the Complex data type an approximate range of 2E-308 to 1E+308 and 15 decimal digits of precision. Both positive and negative numbers are represented. MINREAL and MAXREAL are functions which return the smallest and largest positive real numbers. The range for negative numbers is -MINREAL to -MAXREAL.

Use the COMPLEX statement to declare local complex variables and the COM statement to declare global complex variables and use the static variables to declare local persistent complex variables. Use the ALLOCATE statement to declare a local complex variable which can be DEALLOCATEd dynamically. If a variable is not declared, it will automatically be declared local and real unless CONFIGURE DIM OFF is used.

Definition
Context

A context is a program unit with its own environment, including local variables, which can be called recursively by other contexts, and can pass arguments, either by reference or by value. There are four types of contexts: 1) main context, 2) subprogram context, 3) user defined function, 4) CSUB context.

The main context begins with the first line of the program and ends with the program line containing the "END" statement. The main context is started by a RUN command.

A subprogram context begins with a SUB statement and ends with a SUBEND statement. It is called with a CALL statement and terminates with a SUBEND or SUBEXIT statement. Arguments can be passed to a subprogram.

A user defined function begins with a DEF statement and ends with an FNEND statement. It is called from within a numeric or string expression by referencing its name. It terminates and returns a value with a RETURN statement. The expression then continues to evaluate, using the value returned in place of the function reference. Arguments can be passed to a function.

A CSUB is a compiled subprogram created with special tools outside of HTBasic. It is loaded into memory with the LOADSUB statement and removed from memory with the DELSUB statement. It is called with a CALL statement.

Definition
Device Selector

A device selector is a number which specifies a device. It specifies the interface select code (ISC) to which a device is connected. If more than one device can be connected to that interface (i.e., the GPIB interface), then the address of the device is appended after the ISC. It can be just a primary address or a primary address and several secondary addresses. Each address is specified with two digits; thus 1 is specified as 01. A device selector can be up to 15 digits.

Several examples follow: If a printer has a primary address of 1 and is connected to a GPIB interface with ISC 7, then the device selector for the printer is 701. If an instrument is connected to the RS-232 interface with ISC 9, then the device selector for the instrument is 9. If a GPIB plotter has a primary address of 2, a secondary address of 11 and is connected to a GPIB interface with ISC 14, then the device selector for the plotter is 140211.

Definition
DOS file type

HTBasic supports ordinary files as well as typed files. HTBasic file types are LIF ASCII, BDAT, BIN and PROG. In a CAT listing ordinary files leave the file type column blank. Unlike typed files, no special header or other embedded information is placed in the file. An ordinary file with FORMAT ON is compatible with all programs that support DOS ASCII files. See CREATE in Chapter 4, the "Keyword Dictionary."

Definition
Event

An event is the occurrence of an action or condition which can be trapped by an ON statement that directs program execution to a service routine. See ON in Chapter 4, the "Keyword Dictionary."

Definition
File Specifier

A file specifier identifies a file. Legal file specifiers depend on the operating system and are summarized here. Consult your operating system manuals for complete rules.

A file specifier consists of an optional drive letter, an optional path, a filename and an optional filename extension combined as follows:

```
d:\path\filename.ext
```

The drive letter specifies the disk drive, A, B, C, etc. If it is present, it must be followed by a colon, ":". The path is a series of one or more directory names, separated by the backslash character, "\", leading from the root directory to the file in question. A legal directory name follows the same rules as a legal filename.

The filename consists of 1 to 256 characters, including one or more extensions. Case is ignored by NT although when a new filename is specified, case is preserved for display in a directory listing. Some characters are not legal in a filename. Characters less than CHR$(31) are not legal. The characters in the following list are also illegal: "*/:<>?\|. Trailing spaces are ignored; elsewhere spaces are acceptable.

Definition
Full Array Specifier

A full array specifier is the symbol "(*)" and is used to reference an entire array rather than an individual element.

Definition

Function Name

The rules for naming a function are the same as for a variable (see Variable Name). A User Defined Function is one of several types of contexts (see Context).

Definition
I/O PATH

"I/O path" is a data type. Other data types are integer, long, real, complex and string. An I/O path is implicitly declared whenever you use it in a program. It must be initialized with the ASSIGN statement before it is used. Input and Output statements use an I/O path to specify the entity (device, file, buffer, etc.) that the computer communicates with during the I/O operation. When an input/output statement does not explicitly involve an I/O path, one is created internally, used for the duration of the statement and then discarded.

Definition
Integer

“Integer” is a data type. Other data types are I/O path, real, long, complex, and string. Integers are whole numbers (-1, 35) as opposed to real numbers that can have fractional parts (1.7, 2.34). Integers are stored in two bytes and have a range of -32768 to +32767. Integer operations are faster and integers take less space to store.

Use the INTEGER statement to declare local integer variables, the COM statement to declare global integer variables, and the STATIC statement to declare local persistent variables. Use the ALLOCATE statement to declare a local integer variable which can be DEALLOCATEd dynamically. If a variable is not declared, it will automatically be declared local and real unless CONFIGURE DIM OFF is used.

Definition
Integer Array

Each element of an array (see Array) is an integer declared with INTEGER.

Definition
Interface Select Code

Interface select codes (ISC) specify hardware interfaces that connect the computer to devices. Some ISCs are fixed:

ISC	Fixed Devices
1	CRT display
2	Keyboard
3	Graphic display
6	Bit mapped graphic
10	Windows Print Manager
26	Parallel Port
32	Processor

Others can be specified when the device is loaded with LOAD BIN. If the ISC is not specified, the following defaults are used:

ISC	Loadable Devices
7	GPIB Board
8	2nd GPIB Board
9	RS-232 Port (COM1)
11	2nd RS-232 Port (COM2)
12	GPIO Board
18	Several data acquisition boards

Definition
Line Label

Line labels may optionally follow any line number. The use of line labels results in more structured programming. Line references to labels are unaffected by line numbering. The rules for naming a line label are the same as for variables (see Variable Names). A colon follows the name in the line that is labeled, but does not follow the name in lines referencing that line.

Definition
Line Number

Each program line requires a unique line number at the beginning of the line. Line numbers must be in the range of 1 to 4,194,304. HTBasic ignores leading zeros and spaces before line numbers. Line numbers are used to:

- indicate the order of statement execution
- provide control points for branching
- help in debugging and updating programs
- indicate the location of run-time errors

Definition
Local Variable

All variables are local and are accessible only in the current context unless declared as COM variables. When the context begins execution, storage space is allocated for all local variables and their values are set to zero. When execution of the context is completed, the local variable storage space is released and their values are lost.

Definition

Long

Long is a data type. Other data types are I/O path, integer, real, complex, and string. Longs are whole numbers (-1, 35) as opposed to real numbers that can have fractional parts (1.7, 2.34). Longs are stored in two bytes and have a range of -2,147,483,648 to 2,147,483,647.

Use the LONG statement to declare local integer variables, the COM statement to declare global integer variables, and the STATIC statement to declare local persistent variables. Use the ALLOCATE statement to declare a local long variable which can be DEALLOCATEd dynamically. If a variable is not declared, it will automatically be declared local and real unless CONFIGURE DIM OFF is used.

Definition
Matrix

A matrix is a two dimensional numeric array. The RANK of a matrix is two.

Definition
Numeric Array

A numeric array is an array (see Array) in which the data type of each element is either integer, long, real or complex.

Definition
Numeric Array Element

A numeric array element is a simple value, either an integer, real, or complex number and is compatible with any operation which expects a single value. An element is specified by following the array name with a left parenthesis, "(", a comma-separated list of subscripts and a right parenthesis, ")". The number of subscripts specified must match the RANK of the array. The value of each subscript must lie in the legal range for that dimension as defined in the declaration statement (ALLOCATE, COM, STATIC, COMPLEX, DIM, INTEGER, LONG, REAL, REDIM). Some matrix operations redefine the range of a dimension.

Definition
Numeric Constant

A constant is an entity with a fixed value. There are three types of numeric constants: integer, long and real. An integer constant is a whole number not specified with a decimal point, ".", nor with scientific notation, which falls in the range -2,147,483,647 to 2,147,483,647. Integer constants can be expressed in decimal, octal (base 8) or hexadecimal (base 16). An octal constant must begin with the characters "&O" or simply "&". A hexadecimal constant must begin with the characters "&H. A real constant is specified with a decimal point or scientific notation, or is outside the integer range. Some integer constants are "1", "-20000", "&H7FFF" and "&O377. Some real constants are "-1.0", "1E+10" and "40000.5".

Definition
Numeric Expression

A numeric expression is any legal combination of operands and operators joined together in such a way that the expression as a whole can be reduced to a numeric value. The following syntax diagram defines the legal combination of operands and operators. Precedence rules provide additional constraints on an expression (see Precedence).

```
numeric-expression =
{ + | - | NOT } numeric-expression |
( numeric-expression ) |
numeric-expression operator numeric-expression |
numeric-constant | numeric-name |
numeric-array-element |
numeric-function [ ( param [,param...] ) ] |
FN function-name [ ( param [,param...] ) ] |
string-expression compare-operator string-expression
```

where:

```
operator = + | - | * | / | DIV | MOD | MODULO | ^ |
AND | OR | EXOR | compare-operator
compare-operator =    <> | = | < | > | <= | >=
numeric-function = a function, like COS, which returns a numeric value.
param = legal parameters for numeric functions and user defined
functions are explained in Chapter 4, the "Keyword Dictionary"
```

Definition
Numeric Name

The rules for naming a numeric variable are explained under "Variable Name". A numeric variable is of type integer, long, real or complex.

Definition
Ordinary file

HTBasic supports ordinary files as well as typed files. HTBasic file types are LIF ASCII, BDAT, BIN and PROG. All other files are ordinary files. In a CAT listing, the file type column is blank for ordinary files or gives the operating system (i.e., "DOS" or "HP-UX"). Unlike typed files, no special header or other embedded information is placed in the file. Under Windows, an ordinary file with FORMAT ON is compatible with all programs that support Windows ASCII files. See CREATE in Chapter 4, the "Keyword Dictionary."

Definition
Path Specifier

A path specifier in HTBasic is similar to an MSUS (Mass Storage Unit Specifier) in HP BASIC. It identifies a place where files are stored. Depending on your operating system, the necessary information to uniquely identify such a place includes: the device, address, volume, unit, and directory path list. A summary of the rules is given here. Consult your operating system manuals for complete rules.

A path specifier consists of an optional disk drive letter and an optional directory path. If the disk drive letter is omitted, the default disk is used. A directory path is composed of the names of the directories which form the path from the root directory "\", to the directory where you wish to access files. Each directory name is separated from the others with the backslash, "\", symbol. The rules for each directory name are the same as for a filename (File Specifier). If the directory path is omitted, the default directory is used.

For example, suppose that you wish to use drive "C:" and a catalog of the root directory "C:\" shows a directory named "HTB". Suppose that a catalog of "C:\HTB" shows a directory named "FILES.BIN". And suppose that it is this directory you wish to specify with a path specifier. The correct path specifier is "C:\HTB\FILES.BIN". If drive "C:" is the default drive, then the "C:" could be omitted. If directory HTB is the default directory, then the "\HTB\" could be omitted. Please read your operating system manual for a greater understanding of these concepts.

Definition
Pen Number

The term "pen number" is used in two different ways. The appropriate range is explained in the text describing the statement.

The first way in which the term "pen number" is used is for CRT color attribute values. The legal values are:

Pen	Color
136	White
137	Red
138	Yellow
139	Green
140	Cyan
141	Blue
142	Magenta
143	Black

The second way in which the term "pen number" is used is in statements affecting graphic colors. In these instances, pen numbers begin at zero and go to N-1, where N is the number of colors displayable at the same time on the computer display.

Definition
Pipe Specifier

A pipe specifier is a string beginning and/or ending with the pipe character. Under UNIX, the pipe character is the vertical bar, "|". The remainder of the string specifies one or more processes to be executed. If the pipe-specifier begins with the "|" pipe character, then OUTPUT can be used to send information to the process. If the pipe-specifier ends with the pipe character, then ENTER can be used to get information from the process. Pipes are not supported by HTBasic.

Definition
Precedence

Mathematical precedence describes the order in which operators in an expression are evaluated. Some cheap calculators execute each operation as it is entered. If you are used to this type of calculator, you may be confused by the concept of precedence. For example, the correct answer to the formula:

```
1+2*3+4
```

is 11, not 13. This is because multiplication (2*3) has a higher precedence than addition (1+2). If the two operators are on the same row in the precedence chart, the operations occur in left to right order (i.e. 1+2-3+4).

HP BASIC (and HTBasic) has an odd quirk in its definition of precedence which you should be aware of. Most computer languages place all monadic operators (operators which operate on one operand) at a higher precedence than dyadic operators (operators which operate on two operands). However, HTBasic and HP BASIC place monadic + and - below some of the dyadic operators. The following is one example of an expression that will evaluate differently because of this:

```
-4^0.5
```

With HTBasic, this is equivalent to -(4^0.5) which is equal to -2. With most other computer languages, this is equivalent to (-4)^0.5 which is an illegal operation.

Precedence Table

1	Parentheses () and sub-strings []
2	Functions: built in and user defined.
3	Exponentiation Operator ^
4	Multiplicative Operators *,/,DIV,MODULO,MOD
5	Monadic + and -
6	Dyadic + and -
7	String Concatenation &
8	Relational Operators =,<>,<,>,<=,>=
9	Monadic Logical Operator NOT
10	Logical Operator AND
11	Logical Operators OR and EXOR

Definition
Primary Address

A primary address is a numeric expression which can be rounded to an integer in the range 0 to 31. It specifies the address of a device on the GPIB bus. Usually, GPIB devices have a switch which allows their primary address to be set to any of the values 0 through 31.

Definition
Priority

Priority is a measure of the relative importance of the currently executing line and allows higher priority events to interrupt lower priority events, while preventing lower priority events from interrupting higher priority events. Priority values can range from 0 (least important) to 15 (most important). The ON statement which defines the service routine for an event also allows the priority for that service to be defined. The system priority is the priority of the currently executing line and can be changed with the SYSTEM PRIORITY statement.

Definition
PROG file type

PROG files are used to hold binary program images and are the most efficient file type for storing an HTBasic program. See STORE in Chapter 4, the "Keyword Dictionary" for information about PROG files.

Definition
Real

"Real" is a data type. Other data types are integer, long, complex, string, and I/O path. The Real data type is a subset of all rational numbers. The particular subset depends on your computer. Most computers, including the IBM PC, use IEEE Std 754-1985 for Binary Floating point numbers. This gives the Real data type an approximate range of 2E-308 to 1E+308 and 15 decimal digits of precision. Both positive and negative numbers are represented. MINREAL and MAXREAL are functions which return the smallest and largest positive real numbers. The range for negative numbers is -MINREAL to -MAXREAL.

Use the REAL statement to declare local real variables and the COM statement to declare global real variables and STATIC variables to declare persistent variables. Use the ALLOCATE statement to declare a local real variable which can be DEALLOCATEd dynamically. If a variable is not declared, it will automatically be declared local and real unless CONFIGURE DIM OFF is used.

Please Note: Internally real numbers are represented in a binary format (explained in the *User's Guide*). You need not understand this format, but you should understand its implications. It is possible to have two different numbers in this format whose 15 digit decimal representations are the same. However, when comparing or subtracting these two "look-equal" numbers, you will find they are not equal. Also, when the result of an arithmetic operation is a number not representable in the binary format, an approximation must be used instead. You should take this into account and keep track of the error bounds as approximate numbers are used in further calculations.

Definition
Record Number

The record number is a numeric expression which is rounded to an integer to specify a record within a file. The first record is one. BDAT and ordinary files allow random access by specifying a record number in the I/O statement. The record length for ordinary files is always one. The record length for BDAT files is defined when the file is created with the CREATE BDAT statement.

Definition
Scientific Notation

Scientific notation can be used to represent numbers by using the shorthand notation "n.nnnEmmm" instead of "n.nnn x 10^mmm".

Definition
Signal Number

A signal number is a numeric expression rounded to an integer in the range 0 to 15. A signal is an event which can be generated by the SIGNAL statement and can be handled by a routine set up with the ON SIGNAL statement.

Definition
Softkey Macro

Also called a typing aid, a softkey macro is a sequence of keys assigned to a softkey. When the softkey is pressed, the sequence is typed into the keyboard buffer just as if you had typed them yourself. The definition of the softkey macro is user definable.

Definition

Static

STATIC is a data condition. STATIC variables are persistent during a single run of an HTBasic program. Typically, STATIC variables will only be used in SUB programs and/or FN functions because the MAIN context is usually called only once.

STATIC variables can effectively take the place of COM variables as they are presently used in many cases. If access to a COM variable is required in multiple SUBs and/or Functions (DEF FN) and/or the Main context, then a STATIC variable is not appropriate. The scope of a STATIC variable is limited to the context in which it is declared. In other words, a STATIC variable declared in a SUB program cannot be accessed anywhere other than within that particular SUB program.

Up to 6 bounds may be specified, the initial values are optional. Specifying an initial value for an array initializes each individual element in all dimensions of the array to the initial value specified.

Definition
String

"String" is a data type. Other data types are integer, real, complex, and I/O path. A string is a combination of ASCII characters. These are the letters, numbers and symbols that you can type on the keyboard. ASCII characters also include control characters such as carriage return, etc. A string can be just one character long or it can be one word, one sentence, one paragraph long or any combination of letters, numbers, spaces and symbols up to a maximum length of 32767 characters.

Use the DIM statement to declare a local string variable and define its maximum length. The length of a string variable can never exceed its declared length. Use the ALLOCATE statement to declare a local string variable which can be DEALLOCATEd dynamically. Use the COM statement to declare a global string variable or use the STATIC statement to declare a local persistent STRING variable. If a string variable is not declared, it will be automatically declared as an 18 character maximum length local string variable unless CONFIGURE DIM OFF is used.

Definition
String Array

A string array is an array (see Array) in which the data type of each element is string.

Definition
String Array Element

A string array element is a simple string and is compatible with any function or operation which expects a single string value. An element is specified by following the array name with a left parenthesis, "(", a comma-separated list of subscripts and a right parenthesis, ")". The number of subscripts specified must match the RANK of the array.

Definition
String Expression

A string expression is any legal combination of operands and operators joined together in such a way that the expression as a whole can be reduced to a string value. The following syntax diagram defines the legal combination of operands and operators.

```
string-expression =
( string-expression ) |
string-expression & string-expression |
"string-literal" |
string-name |
string-array-element |
sub-string |
string-function [ ( param [,param...] ) ] |
FN function-name$ [ ( param [,param...] ) ]
```

where:

string-function = a function, like UPC$, which returns a string value.
param = legal parameters for string functions and user defined
functions are explained in Chapter 4, the "Keyword Dictionary."

Definition
String Literal

A string literal is a string of characters delimited by the quote (") character. To include a quote character in the string, include two quote characters in the place of the one you wish to include. For example " ""hello"" ".

Definition
String Name

The rules for naming a string variable are the same as for a variable (see Variable Name) plus the addition of a trailing dollar sign, "$". A string variable is a variable whose data type is "string".

Definition
Sub-string

A substring defines a portion of a string variable or string array element. It is selected by specifying a starting position within the string value and optionally, either the length of the sub-string, or the ending position within the string value. If only the starting position is specified, the rest of the string value from that point on is used for the sub-string. String positions are one-based, i.e., the first character of a string is in position one. The syntax is as follows:

```
sub-string =
'[' start-pos ']' |
'[' start-pos, end-pos ']' |
'[' start-pos; length ']'
```

where:

> start-pos and end-pos = numeric expression rounded to an integer in the range 1 to 32767.
> length = numeric expression rounded to an integer
> in the range 0 to 32767.

Definition
Subprogram Name

The rules for naming a subprogram are the same as for a variable (see Variable Name). A subprogram is one type of context (see Context).

Definition
Subscript

A subscript is a numeric expression rounded to an integer to specify an array dimension. The value of each subscript must lie in the legal range for that dimension as defined in the declaring statement (ALLOCATE, COM, COMPLEX, DIM, INTEGER, LONG, REAL, REDIM). Some matrix operations automatically redefine the range of a dimension.

Definition
UNIX file type

HTBasic supports ordinary files as well as typed files. HTBasic file types are LIF ASCII, BDAT, BIN and PROG. In a CAT listing, the file type column is blank for ordinary files or gives the operating system (i.e., "DOS" or "HP-UX"). Unlike typed files, no special header or other embedded information is placed in the file. See CREATE in Chapter 4, the "Keyword Dictionary."

Definition
Variable Name

A variable name can have up to fifteen characters. The characters can be alphabetic, numerals, underlines and characters in the range CHR$(128) to CHR$(254). (HP BASIC and some versions of HTBasic use the range CHR$(161) to CHR$(254).) The first character may not be a numeral or an underline. A variable name can be the same as a keyword if it is entered partly in upper case and partly in lower case. Variable names are listed with the first character in upper case and the remaining characters in lower case.

Definition

Vector

A vector is a one dimensional numeric array, i.e., the RANK of the array is one.

Definition
Volume Label

A volume label is present in some operating systems to label a mass storage volume (usually a disk). The rules for legal volume labels differ from system to system.

Definition
Volume Specifier

A volume specifier in HTBasic is similar to an MSUS (Mass Storage Unit Specifier) in HP BASIC. However, for disk volumes with multiple directories, a volume specifier does not completely identify a place to store files (see Path Specifier).

Two types of volume specifiers are supported by HTBasic. The first is the native type used by your operating system. For Windows, a volume specifier is the drive letter followed by a colon. For example, "C:". If used with a file specifier, it is appended onto the front of the filename, "C:DATA". For other operating systems, consult your manuals.

The second type of volume specifier supported by HTBasic is the HP BASIC compatible msus style. For example, ":CS80,700,0". Support for this type is included for compatibility with old HP programs. To use this type of volume specifier you must use the CONFIGURE MSI statement to define a translation between this type of volume specifier and the native type used by your system. For example:

```
CONFIGURE MSI ":CS80,700,0" TO "B:"
CONFIGURE MSI ":A" TO "A:"
CONFIGURE MSI ":,1400,1" TO "C:\HTB\1400\1"
```

The first example would allow a file specifier such as "DATA:CS80,700,0". The second example would allow a file specifier such as "DATA:A". If the CONFIGURE statement is not used, then an HP BASIC style volume specifier will cause an error. The third example shows an HP style volume specifier being equated with a Windows style path specifier.

Keywords

A

ABORT	Stops IEEE-488 activity.
ABORTIO	Stops an active TRANSFER.
ABS	Returns the absolute value of an expression.
ACS	Returns the arccosine of an expression.
ACSH	Returns the hyperbolic arccosine of an expression.
ALLOCATE	Dynamically allocates memory for string variables and arrays.
ALPHA	Controls the visibility of the ALPHA screen area.
ALPHA HEIGHT	Sets the number of lines used for the ALPHA screen.
ALPHA PEN	Sets the ALPHA display color.
AND	Performs the logical conjunction of two expressions.
APPEND	See ASSIGN, DUMP DEVICE IS, PLOTTER IS, PRINTALL IS and PRINTER IS.
AREA	Sets or defines an AREA fill color.
ARG	Returns the Argument (Angle) of a complex number.
ASCII	See CREATE ASCII and LEXICAL ORDER IS.
ASN	Returns the arcsine of an expression.
ASNH	Returns the hyperbolic arcsine of an expression.
ASSIGN	Sets up an I/O path and its attributes.
ATN	Returns the arctangent of an expression.
ATNH	Returns the hyperbolic arctangent of an expression.
ATN2	Returns the angle to a point.
AXES	Draws x-y axes.

Keywords B

BASE	Returns the lower bound of an array dimension.
BDAT	See CREATE BDAT and CONFIGURE BDAT.
BEEP	Generates music or sound effects.
BIN	See LIST BIN, LOAD BIN and SCRATCH.
BINAND	Performs a bit by bit logical AND.
BINCMP	Performs a bit by bit complement.
BINEOR	Performs a bit by bit exclusive OR (EXOR).
BINEQV	Performs a bit by bit equivalence operation.
BINIMP	Performs a bit by bit implication operation.
BINIOR	Performs a bit by bit inclusive OR.
BIT	Allows any bit in an INTEGER to be tested.
BREAK	Sends a BREAK on a serial interface.
BUFFER	See ASSIGN, COM, DEF FN, DIM, INTEGER, REAL and SUB.
BYTE	See ASSIGN.

Keywords C

CALL	Starts execution at the specified SUBprogram or CSUB.
CASE	See SELECT ... CASE.
CAT	Displays a catalog of files or PROG file contexts.
CAUSE ERROR	Simulates a specified error.
CD	See MASS STORAGE IS.
CHANGE	Finds and replaces strings.
CHECKREAD	Enables/disables verification of data sent to disk.
CHGRP	Sets the Group Ownership of a file.
CHOWN	Sets the Individual Ownership of a file.
CHR$	Creates an ASCII character from its decimal numeric code.
CHRX	Returns the width of a character cell.
CHRY	Returns the height of a character cell.
CINT	Converts a value to INTEGER.
CLEAR	Sends an IEEE-488 bus Device Clear.
CLEAR ERROR	Resets all error indicators.
CLEAR LINE	Clears the keyboard input line.
CLEAR SCREEN	Clears the ALPHA display.
CLIP	Changes the clipping rectangle.
CLS	See Also CLEAR SCREEN.
CMD	See SEND.
CMPLX	Combines real and imaginary parts to return a complex number.
COLOR	Defines and selects the color for graphics.
COM	Defines global variables.
COMMAND$	Returns a copy of the command line.
COMPLEX	Reserves storage for complex variables and arrays.
CONFIGURE BDAT	Specifies the byte order for CREATE BDAT.
CONFIGURE CREATE	Specifies the kind of file header used with typed files.
CONFIGURE DIM	Turns implicit variable dimensioning on or off.
CONFIGURE DUMP	Specifies what graphic printer language to use for DUMP.
CONFIGURE KBD	Defines keyboard mappings for character sets.
CONFIGURE KEY	Assigns editor functions to keyboard keys.
CONFIGURE LABEL	Defines characters for the LABEL statement.
CONFIGURE LONGFILENAMES	Specifies use of long filenames.
CONFIGURE MSI	Specifies HP style volume specifier translations.
CONFIGURE PRT	Specifies the value of PRT.
CONFIGURE SAVE	Sets the file type produced by SAVE.
CONJG	Returns the conjugate of a complex number.
CONT	Restarts a program which is PAUSEd.
CONTROL	Sends control information to an interface or I/O path.
CONVERT	This ASSIGN option is not supported.
COPY	Copies files.
COPYLINES	Copies one or more program lines from one location to another.
COS	Returns the cosine of an expression.
COSH	Returns the hyperbolic cosine of an expression.
COUNT	See CAT.

CREATE	Creates an ordinary file on the mass storage media.
CREATE ASCII	Creates a LIF ASCII file on the mass storage media.
CREATE BDAT	Creates a BDAT (binary data) file on the mass storage media.
CREATE DIR	Creates directories on the mass storage media.
CRT	Returns the integer 1, the CRT interface select code.
CSIZE	Sets the character size for LABEL and SYMBOL.
CSUB	Compiled SUBprograms.
CSUM	See MAT.
CVT$	Convert strings from one alphabet to another.
CYCLE	See OFF CYCLE and ON CYCLE.

Keywords D

DATA	Stores data items in the program.
DATE	Converts a string representing a date to a number of seconds.
DATE$	Takes a numeric value representing seconds and formats it into a date string.
DEALLOCATE	Frees memory space reserved by the ALLOCATE statement.
DEF FN	Begins a user-defined function subprogram.
DEG	Sets the trigonometric mode to degrees.
DEL	Deletes program lines.
DELAY	See ASSIGN, OFF DELAY, ON DELAY, PRINTALL IS and PRINTER IS.
DELSUB	Deletes SUB or CSUB subprograms from memory.
DET	Returns the determinant of a matrix.
DIGITIZE	Inputs digitized X and Y coordinates.
DIM	Dimensions REAL arrays and strings.
DISABLE	Disables event-initiated branches.
DISABLE INTR	Disables interrupts from the specified interface.
DISP	Displays items on the CRT display line.
DISPLAY FUNCTIONS	Controls the display of control characters on the CRT.
DIV	Returns the quotient of an integer divide operation.
DOT	Returns the dot product of two numeric vectors.
DRAW	Draws a line to the X,Y location.
DROUND	Rounds a numeric-expression to the specified number of digits.
DUMP	Copies the contents of the display to a printing device.
DUMP DEVICE IS	Defines the printing device used by DUMP.
DVAL	Converts a binary, octal, decimal or hexadecimal string to a real number.
DVAL$	Converts a number to a binary, octal, decimal or hexadecimal string.

Keywords E

ECHO	See SET ECHO.
EDGE	See IPLOT,PLOT,POLYGON,RECTANGLE,RPLOT and SYMBOL.
EDIT	Puts you into program EDIT mode.
EDIT KEY	Puts you into softkey EDIT mode.
ELSE	See IF ... THEN and SELECT ... CASE.
ENABLE	Enables all event-initiated branches suspended by DISABLE.
ENABLE INTR	Enables interrupts from a specified interface.
END	Marks the end of the program.
END IF	See IF ... THEN.
END LOOP	See LOOP.
END SELECT	See SELECT ... CASE.
END WHILE	See WHILE.
ENTER	Inputs data and assigns it to variables.
ENVIRON$	Returns information from the operating system environment.
EOL	See ASSIGN,PRINTALL IS and PRINTER IS.
ERRDS	This function is not supported.
ERRL	Compares a line number with ERRLN.
ERRLN	Returns the program line number on which the last error occurred.
ERRM$	Returns the error message text of the last error.
ERRN	Returns the last error number.
ERROR	See CAUSE ERROR,CLEAR ERROR,ERROR RETURN,ERROR SUBEXIT,OFF ERROR,ON ERROR.
ERROR RETURN	Returns program execution to the line following the most recent error.
ERROR SUBEXIT	Returns subprogram execution to the line following the most recent error.
EXECUTE	Executes an operating system command.
EXIT IF	See LOOP.
EXOR	Performs a Logical exclusive OR of two expressions.
EXP	Returns "e" raised to a power.
EXPANDED	See DUMP DEVICE IS.

Keywords F

FBYTE	Determines if character is first byte of a two byte character.
FILL	See IPLOT,PLOT,POLYGON,RECTANGLE,RPLOT and SYMBOL.
FIND	Searches for specified characters in a program.
FIX	Truncates a value to INTEGER.
FN	Executes a user-defined function.
FNEND	Ends a function definition. See DEF FN.
FOR ... NEXT	Executes a loop a fixed number of times.
FORMAT	See ASSIGN.
FRACT	Returns the fractional part of an argument.
FRAME	Draws a frame around the clipping area.
FRE	Returns the amount of free memory.
FRENCH	See LEXICAL ORDER IS.
FROM	See LOADSUB and READ LABEL.

Keywords G

GCLEAR	Clears the graphics screen.
GERMAN	See LEXICAL ORDER IS.
GESCAPE	Sends device-specific information to a graphic device.
GET	Loads LIF, DOS, UNIX, Viper-I and Viper-II ASCII program file into memory.
GINIT	Initializes graphics parameters to their default values.
GLOAD	Loads an integer array into the CRT display buffer.
GOSUB	Transfers control to a subroutine.
GOTO	Transfers control to a specified line.
GRAPHICS	Makes the graphics screen visible or invisible.
GRAPHICS INPUT IS	Defines the device to be used for graphic input.
GRID	Draws a grid pattern.
GSEND	Sends commands to the PLOTTER IS device.
GSTORE	Stores the CRT display buffer into an integer array.

Keywords H

HELP	Outputs *Reference Manual* pages to the computer screen.

Keywords I

IDN	See MAT.
IDRAW	Draws a line an incremental distance.
IF ... THEN	Performs an action if a condition is true.
IMAG	Returns the imaginary part of a complex number.
IMAGE	Defines the format for data input and output.
IMOVE	Lifts and moves the logical pen position incrementally.
INDENT	Indents a program to reflect its structure.
INITIALIZE	Initializes the mass storage media for use by the computer.
INMEM	Identifies if a subprogram is loaded.
INP and INPW	Inputs a byte or word from an I/O Port.
INPUT	Inputs numeric or string data from the keyboard.
INPW	See INP.
INT	Performs the greatest integer function.
INTEGER	Declares, dimensions and reserves memory for INTEGER variables.
INTENSITY	See AREA,COLOR and SET PEN.
INTERACTIVE	See RESUME INTERACTIVE and SUSPEND INTERACTIVE.
INV	See MAT.
IPLOT	Moves the pen relative to its present location.
IVAL	Converts a binary, octal, decimal or hexadecimal string to an INTEGER.
IVAL$	Converts an INTEGER to a binary, octal, decimal or hexadecimal string.

Keywords K

KBD	Returns a 2, the device select code of the keyboard.
KBD$	Returns the contents of the ON KBD buffer.
KBD CMODE	Sets softkey compatibility mode.
KBD LINE PEN	Sets the pen color for the input line.
KEY LABELS	Controls the display of the softkey labels.
KEY LABELS PEN	Sets the color for the softkey labels.
KEY	See CONFIGURE KEY,EDIT KEY,LIST KEY,LOAD KEY,OFF KEY,ON KEY, READ KEY,SCRATCH,SET KEY and STORE KEY.
KNOB	See OFF KNOB and ON KNOB.
KNOBX	Returns and resets the KNOBX counter value.
KNOBY	Returns and resets the KNOBY counter value.

Keywords L

LABEL	Prints text on graphic devices.
LDIR	Sets the angle for drawing LABELs and SYMBOLs.
LEN	Returns the number of characters in a string.
LET	Assigns a value to a variable.
LEXICAL ORDER IS	Defines "alphabetical" order for string comparisons.
LGT	Computes common (base 10) logarithms.
LINE TYPE	Sets the style or dash pattern and repeat length of lines.
LINK	Makes a hard link to a file.
LINPUT	Assigns alphanumeric keyboard input to a string variable.
LIST	Lists the program in memory to the selected device.
LIST BIN	Lists each BIN currently in memory.
LIST KEY	Lists the softkey macro definitions.
LISTEN	See SEND.
LOAD	Loads a user program into memory.
LOAD BIN	Loads a BIN system program file into memory.
LOAD KEY	Loads softkey macro definitions into memory.
LOADSUB	Loads a BASIC subprogram into memory.
LOCAL	Returns specified IEEE-488 devices to their local state.
LOCAL LOCKOUT	Sends the IEEE-488 LLO message.
LOCATOR	See READ LOCATOR and SET LOCATOR.
LOCK	Secures a file for exclusive access.
LOG	Computes natural (base "e") logarithms.
LOOP	Defines a series of statements to be executed repeatedly.
LORG	Specifies the position of a LABEL relative to the current position.
LWC$	Converts characters in a string to lowercase.

Keywords M

MASS STORAGE IS	Assigns the current mass storage device and directory.
MAT	Specifies an array operation.
MAT REORDER	Reorders array elements by a supplied subscript list.
MAT SEARCH	Searches an array for user specified conditions.
MAT SORT	Sorts string or numeric array data.
MAX	Returns the maximum value of a list of expressions.
MAXLEN	Returns the maximum declared length of a string variable.
MAXREAL	Returns the largest positive REAL number.
MERGE ALPHA	Enables all planes for Alpha and Graphics.
MIN	Returns the minimum value of a list of expressions.
MINREAL	Returns the smallest positive REAL number.
MLA	See SEND.
MOD	Returns the remainder after integer division.
MODULO	Returns the true mathematical modulus.
MOVE	Moves the logical and physical pens to a new position.
MOVELINES	Moves one or more program lines from one location to another.
MSI	See MASS STORAGE IS.
MTA	See SEND.

Keywords
N

NEXT	See FOR.
NOT	Returns the logical negation of an expression.
NPAR	Returns the number of parameters passed to a subprogram.
NUM	Returns the decimal ASCII equivalent of the first character in a string.

Keywords O

OFF	See ALPHA OFF, CLIP OFF, GRAPHICS OFF, TRACE OFF.
OFF CYCLE	Cancels event branches defined by ON CYCLE.
OFF DELAY	Cancels event branches defined by ON DELAY.
OFF END	Cancels event branches defined by ON END.
OFF EOR	Cancels event branches defined by ON EOR.
OFF EOT	Cancels event branches defined by ON EOT.
OFF ERROR	Cancels event branches defined by ON ERROR.
OFF INTR	Cancels event branches defined by ON INTR.
OFF KBD	Cancels event branches defined by ON KBD.
OFF KEY	Cancels event branches defined by ON KEY.
OFF KNOB	Cancels event branches defined by ON KNOB.
OFF SIGNAL	Cancels event branches defined by ON SIGNAL.
OFF TIME	Cancels event branches defined by ON TIME.
OFF TIMEOUT	Cancels event branches defined by ON TIMEOUT.
ON	Transfers control to one of a list of lines.
ON	See ALPHA ON, CLIP ON, GRAPHICS ON
ON CYCLE	Defines a repeating event branch.
ON DELAY	Defines an event branch after specified seconds.
ON END	Defines an event branch for end-of-file conditions.
ON EOR	Defines an event branch for end-of-record conditions.
ON EOT	Defines an event branch for end-of-transfer conditions.
ON ERROR	Defines an event branch for trappable errors.
ON INTR	Defines a hardware interrupt initiated branch.
ON KBD	Defines an event branch for when a key is pressed.
ON KEY	Defines an event branch for when a softkey is pressed.
ON KNOB	Defines an event branch for when the KNOB is turned.
ON SIGNAL	Defines an event branch for SIGNAL statement.
ON TIME	Defines a single event branch for a specific time.
ON TIMEOUT	Defines an event branch for an I/O timeout.
OPTION BASE	Sets the default lower bound of array subscripts.
OPTIONAL	See DEF FN and SUB.
OR	Returns the logical inclusive OR of two expressions.
OUT and OUTW	Outputs a byte or word to an I/O Port.
OUTPUT	Outputs items to a specified destination.
OUTW	See OUT.

Keywords
P

PARITY	This ASSIGN option is not supported.
PASS CONTROL	Passes Active Controller capability.
PAUSE	Pauses program execution.
PDIR	Sets the rotation angle for IPLOT, RPLOT, POLYGON and RECTANGLE.
PEN	Sets the line color or physical pen.
PENUP	Raises the PEN on the current plotting device.
PERMIT	Changes file protection permissions.
PI	Returns the value 3.141 592 653 589 79.
PIVOT	Rotates the coordinates of all drawn lines.
PLOT	Moves the pen to the specified X and Y coordinates.
PLOTTER IS	Specifies the graphics output device and language.
POLYGON	Draws a closed regular polygon, circle, or ellipse.
POLYLINE	Draws an open regular polygon.
POS	Returns the position of one string within another.
PPOLL	Conducts a Parallel Poll of the IEEE-488 and returns status.
PPOLL CONFIGURE	Configures remote IEEE-488 device parallel poll response.
PPOLL RESPONSE	Configures local IEEE-488 device parallel poll response.
PPOLL UNCONFIGURE	Disables the parallel poll response of a specified device or devices.
PRINT	Outputs data to the PRINTER IS device.
PRINT LABEL	Assigns a name to a data storage volume.
PRINT PEN	Selects the pen color used for the output area and DISP line.
PRINTALL IS	Assigns a logging device for operator interaction and error messages.
PRINTER IS	Specifies the system printing device.
PRIORITY	See SYSTEM PRIORITY.
PROTECT	Changes file attributes.
PROUND	Rounds the argument to the specified power of ten.
PRT	Returns the default device selector for the printer.
PURGE	Deletes a file or a directory on a mass storage media.

Keywords Q

QUIT	Quits BASIC and returns to the operating system.

Keywords R

RAD	Sets the trigonometric mode to radians for all angle measurements.
RANDOMIZE	Selects a seed for the RND function.
RANK	Returns the number of dimensions in an array.
RATIO	Returns the ratio of X to Y hard-clip limits for the PLOTTER IS device.
READ	Reads values from DATA statements.
READ KEY	Returns one or more softkey macro definitions.
READ LABEL	Reads a volume label.
READ LOCATOR	Reads the locator device without waiting for a digitize operation.
READIO	Reads a hardware register or a memory byte/word.
REAL	Reserves storage for floating point variables and arrays.
REAL	Converts an INTEGER or COMPLEX number to REAL.
RECOVER	See ON-event statements.
RECTANGLE	Draws and optionally fills and edges rectangles.
REDIM	Redimensions an array by changing the subscript ranges.
REM	Begins a REMark or comment line for program documentation.
REMOTE	Sets the remote state on a IEEE-488 device.
REN	Renumbers program lines.
RENAME	Changes the name of a file.
REORDER	See MAT REORDER.
REPEAT ... UNTIL	Defines a loop that is repeated UNTIL a condition is satisfied.
REQUEST	Sends a Service Request SRQ on the IEEE-488.
RE-SAVE	Copies the program into the specified ASCII file.
RES	Returns the result of the last numeric keyboard calculation.
RESET	Resets an interface or file or buffer pointers.
RESET	See SUSPEND INTERACTIVE.
RESTORE	Specifies which DATA statement to use for the next READ operation.
RE-STORE	Stores the BASIC program in a file.
RE-STORE KEY	Stores the KEY definitions in a file.
RESUME INTERACTIVE	Restores the normal functions of program control keys.
RETURN	Returns to the program line following the last GOSUB line.
REV$	Reverses the sequence of characters in a string.
RND	Returns a pseudo-random number.
ROTATE	Shifts a 16 bit binary value with wraparound.
RPLOT	Moves the pen relative to the current graphic location.
RPT$	Returns a string replicated a specified number of times.
RSUM	See MAT.
RUN	Starts program execution.
RUNLIGHT	Controls the display of the pseudo runlight on the display.

Keywords S

SAVE	Saves the current program into an ASCII file.
SBYTE	Determines if character is second byte of a two byte character.
SC	Returns the interface select code associated with an I/O path name.
SCRATCH	Clears user memory.
SEC	See SEND.
SECURE	Protects programs lines.
SELECT ... CASE	Defines a CASE block structure.
SEND	Sends messages on the IEEE-488 bus.
SEPARATE ALPHA	On a bit-mapped display, simulates 9836 style alpha/graphics hardware.
SET ALPHA MASK	Determines which plane(s) can be modified by ALPHA display operations.
SET CHR	Defines the bit-patterns for one or more characters.
SET DISPLAY MASK	Specifies which planes can be seen on the alpha display.
SET ECHO	Sets the echo location on the PLOTTER IS device.
SET KEY	Defines one or more softkey macros.
SET LOCATOR	Sets a new graphic locator position on the GRAPHICS INPUT IS device.
SET PEN	Defines part or all of the color map.
SET TIME	Sets the time of day clock.
SET TIMEDATE	Sets the date and time of the computer's clock.
SGN	Returns the arithmetic sign of an expression.
SHIFT	Shifts a 16 bit binary value.
SHOW	Defines the graphics unit-of-measure isotropically.
SIGNAL	Initiates a software interrupt.
SIN	Returns the sine of the argument.
SINH	Returns the hyperbolic sine of an expression.
SIZE	Returns the number of elements of a dimension of an array.
SORT	See MAT SORT.
SOUND	Produces tones on the computer speaker.
SPANISH	See LEXICAL ORDER IS.
SPOLL	Performs a serial poll of a IEEE-488 device.
SQR	See SQRT.
SQRT	Returns the square root of an expression.
STANDARD	See LEXICAL ORDER IS.
STATUS	Returns control information from an interface or I/O path.
STEP	See FOR.
STOP	Terminates program execution.
STORE	Stores the BASIC program in a file.
STORE KEY	Stores the softkey definitions in a file.
STORE SYSTEM	Stores BASIC and loaded BINs into a file.
SUB	Defines a subprogram and specifies formal parameters.
SUBEND and SUBEXIT	See SUB.
SUM	Returns the sum of all elements in a numeric array.
SUSPEND INTERACTIVE	Deactivates program control keys.
SWEDISH	See LEXICAL ORDER IS.
SYMBOL	Allows the user to define symbols that may be used as labels.
SYSBOOT	Reboots the computer.

SYSTEM KEYS	Displays the System Softkeys Menu.
SYSTEM PRIORITY	Sets the system priority to a specified level.
SYSTEM$	Returns system status and configuration information.

Keywords T

TAB	See DISP and PRINT.
TABXY	See PRINT.
TALK	See SEND.
TAN	Returns the tangent of an expression.
TANH	Returns the hyperbolic tangent of an expression.
THEN	See IF ... THEN.
TIME	Converts a time-of-day string to seconds after midnight.
TIME$	Returns a formatted time of day string.
TIMEDATE	Returns the current time and date from the clock.
TIMEOUT	See OFF TIMEOUT and ON TIMEOUT.
TIMEZONE IS	Corrects between GMT and local time for HP BASIC/WS.
TO	See COPY, COPYLINES, FOR, MAT SORT, MOVELINES, RENAME, SELECT ... CASE.
TRACE	Controls the display of information about a running program.
TRACK	Enables or disables tracking of the locator position on the display device.
TRANSFER	Performs an unformatted I/O transfer.
TRIGGER	Sends a trigger message to all or selected devices on the IEEE-488.
TRIM$	Removes leading and trailing spaces from a string.
TRN	See MAT.

Keywords U

UNL	See SEND.
UNLOCK	Removes exclusive access protection from a LOCKed file.
UNT	See SEND.
UNTIL	See REPEAT.
UPC$	Converts characters in a string to uppercase characters.
USER KEYS	Displays the specified User Softkey Menu.
USING	See IMAGE, ENTER, LABEL, OUTPUT, PRINT.

Keywords V

VAL	Converts a string into a numeric value.
VAL$	Converts a number into its string representation.
VIEWPORT	Defines the area of the graphic device used for output.

Keywords W

WAIT	Waits a specified time or for TRANSFER events.
WHERE	Returns the logical pen position.
WHILE	Repeats an action while a condition is true.
WIDTH	See PRINTALL IS and PRINTER IS.
WILDCARDS	Enables or disables wildcard support.
WINDOW	Sets the bounds for displayable graphics data in user defined units.
WORD	See ASSIGN.
WRITEIO	Writes to a hardware register or a memory byte/word.

Keywords X

XREF	Generates a cross reference of a program.

Keywords Z

ZERO	This ASSIGN option is not supported.

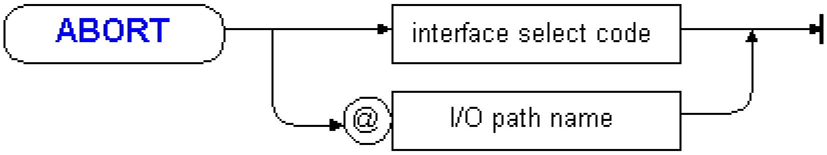
Chapter 4
Keyword Dictionary

The following pages contain the HTBasic keywords listed in dictionary fashion. Each entry includes a syntax diagram, sample statements, a description of the keyword's functionality and related keywords.

ABORT

Stops IEEE-488 activity.

Syntax: ABORT { interface-select-code | @io-path }



Sample:

```
ABORT 7
ABORT Isc
ABORT @Code
```

View Sample: **ABORT.BAS (also found in examples directory)**

Description:

This command is only legal on the IEEE-488 interface. If the computer is the system controller but not the active controller, ABORT causes the computer to assume active control.

If a primary address is specified, an error is generated. If the computer is the system controller, the bus action is to issue IFC for greater than 100 micro-seconds and then to assert REN and de-assert ATN. If the computer is not the system controller but is the active controller, the bus action is: ATN, MTA, UNL and de-assert ATN. If it is also not the active controller, no action is taken.

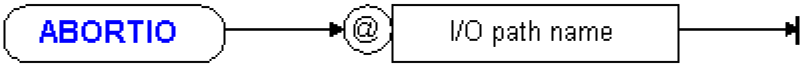
See Also:

CLEAR , LOCAL , PASS CONTROL , PPOLL , REMOTE , REQUEST , SEND , SPOLL , TRIGGER

ABORTIO

Stops an active TRANSFER.

Syntax: ABORTIO @io-path



Sample:

```
ABORTIO @Isc
ABORTIO @Device
```

View Sample: **ABORTIO.BAS** **(also found in examples directory)**

Description:

The I/O must be assigned to an interface select code or device selector, not the BUFFER. If an ON EOT branch is enabled, it will be called. If there is no active TRANSFER on the I/O, then **ABORTIO** has no effect. If a TRANSFER was stopped because of an error, **ABORTIO** report, the error.

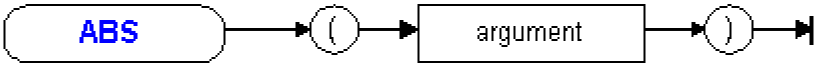
See Also:

BREAK, ON EOR, ON EOT, RESET, TRANSFER, WAIT

ABS

Returns the absolute value of an expression.

Syntax: ABS(numeric-expression)



Sample:

```
J=ABS(X*5)
PRINT "Total losses=";ABS(Sum)
R=ABS(SIN(Theta))
```

View Sample: **ABS.BAS (also found in examples directory)**

Description:

For REAL, INTEGER,and LONG arguments, the result of the **ABS** f unction is the same type as the argument.

COMPLEX Arguments

For COMPLEX arguments, **ABS** returns the absolute value (magnitude or modulus) of the argument. The absolute value of a number CMPLX(X,Y) is the distance from the origin to the point (X,Y) in the complex plane:

ABS(CMPLX(X,Y)) = SQRT(X^2+Y^2)

Notice that intermediate values generated during the calculation of the function can cause over or underflow errors for very large or small values of X and Y. Complex numbers are stored in rectangular form, but may be used in polar form using **ABS** and ARG. For example:

```
PRINT "Magnitude = ";ABS(Z),"Angle = ";ARG(Z)
```

To enter a number in polar form, convert it from polar form to rectangular in this manner:

```
10  COMPLEX Z
20  INPUT Magnitude,Angle
30  Z=CMPLX( Magnitude*COS(Angle), Magnitude*SIN(Angle))
```

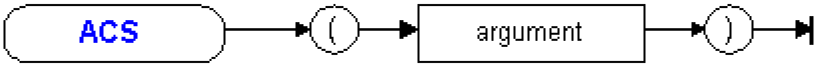
See Also:

ARG, FRACT, INT, SGN

ACS

Returns the arccosine of an expression.

Syntax: ACS(numeric-expression)



Sample:

```
Alpha=ACS(R0)
Angle=ACS(Cosine)
PRINT "Angle = ";ACS(Z)
```

View Sample: **ACS.BAS (also found in examples directory)**

Description:

ACS returns the arccosine of a numeric expression whose value is between -1 and +1 inclusive. The arccosine of a number is the angle whose cosine is that number. **ACS** returns a value between 0 and PI radians or 180 degrees, depending on the current trigonometric mode. The default trigonometric mode is radians.

COMPLEX Arguments

ACS accepts either a COMPLEX or REAL argument and returns a value of the same type. For COMPLEX arguments the angle is returned in radians, regardless of the current trigonometric mode. **ACS** returns the principal value, defined (in terms of complex arithmetic) as

ACS(Z) = CMPLX(0,-1)*LOG(Z+CMPLX(0,1)*SQRT(1-Z^2))

which returns a real part in the range 0 to PI. The domain for COMPLEX arguments includes all points in the complex plane (but for REAL arguments, the domain is still -1 to +1 inclusive). Notice that intermediate values generated during the calculation of the function can cause over or underflow errors for very large or small values of Z.

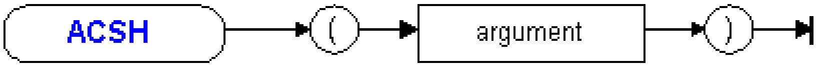
See Also:

ASN, ATN, COS, SIN, TAN, ASNH, ACSH, ATNH, COSH, SINH, TANH, DEG, PI, RAD

ACSH

Returns the hyperbolic arccosine of an expression.

Syntax: ACSH(numeric-expression)



Sample:

```
Angle=ACSH(Hcosine)
PRINT "Complex Angle = ";ACSH(Z)
```

View Sample: **ACSH.BAS (also found in examples directory)**

Description:

The hyperbolic arccosine of a number is the angle whose hyperbolic cosine is that number. The angle is returned in radians, regardless of the current trigonometric mode. **ACSH** returns the principal value, defined (in terms of complex arithmetic) as

ACSH(Z) = LOG(Z+CMPLX(0,1)*SQRT(1-Z^2))

which returns an imaginary part in the range 0 to PI. **ACSH** accepts either a COMPLEX or REAL argument and returns a value of the same type. The domain for COMPLEX arguments includes all points in the complex plane, but for REAL arguments, the domain is only defined for points ≥ 1. Notice that intermediate values generated during the calculation of the function can cause over or underflow errors for very large or small values of Z.

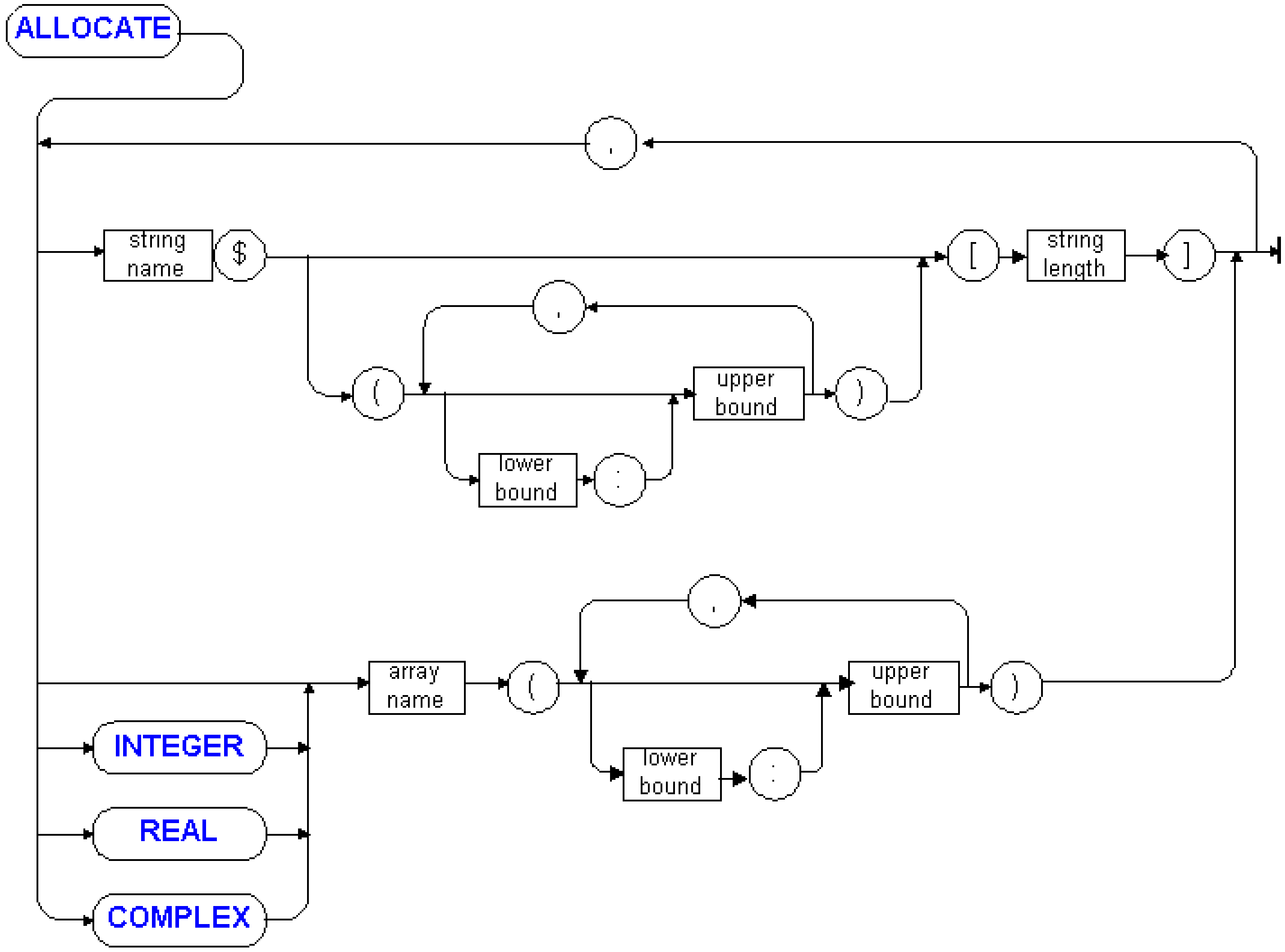
See Also:

ACS, ASNH, ATNH, COSH, SINH, TANH

ALLOCATE

Dynamically allocates memory for string variables and arrays.

Syntax: ALLOCATE item [,item...]



where: item = [type] numeric-array (bounds) |
variable-name$ [(bounds)] '['length']'
type = REAL | INTEGER | COMPLEX
bounds = [lower-bound:] upper-bound [,bounds...]

Sample:

```
ALLOCATE Chart(Down:Up)
ALLOCATE M$[LEN(N$)+1]
ALLOCATE Group$(Section)[50]
ALLOCATE INTEGER Myarray(Type,3,5)
```

View Sample: **ALLOCATE.BAS** **(also found in examples directory)**

Description:

The lower and upper bound range is -32,768 through +32,767, with the default lower bound range being the OPTION BASE (0 or 1). The string length is a numeric expression rounded to an integer in the range of 1 through 32,767.

ALLOCATE variables cannot appear in COM, COMPLEX, DIM, INTEGER or REAL declaration statements or be declared in the subprogram parameter list.

DEALLOCATE frees allocated memory, but because of stack requirements the freed memory does not become available unless all allocated items are also deallocated. In addition, ON event statements also use the stack and will not allow the deallocated memory to be available for use until the ON event statements are released from the stack. Memory **ALLOCATE**d within a subprogram is DEALLOCATEd upon exit of that subprogram.

After a variable has been deallocated, it can be reallocated with a different size as long as it has the same type and number of dimensions.

Porting Issues

Under HTBasic, GOSUB and **ALLOCATE** use the same stack. Intermixing these statements can cause changes in available memory that are different from HP BASIC. This usually does not cause problems.

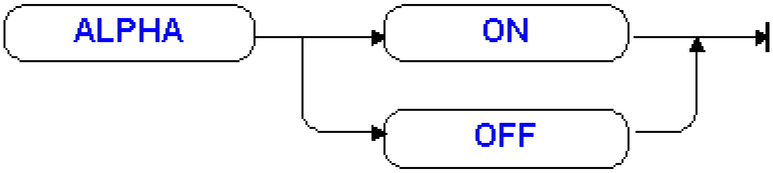
See Also:

COM, COMPLEX, DEALLOCATE, DIM, INTEGER, OPTION BASE, REAL, REDIM

ALPHA

Controls the visibility of the ALPHA screen area.

Syntax: ALPHA { ON | OFF }



Sample: `ALPHA ON`

```
IF Display THEN ALPHA OFF
```

Description:

ALPHA ON makes the alpha screen visible; **ALPHA OFF** makes it invisible. The current screen driver has an effect on the execution of this statement as explained in the following paragraphs. See PLOTTER IS for an explanation of the screen drivers.

ALPHA ON/OFF has no effect when **ALPHA** and GRAPHICS are MERGEd. SEPARATE ALPHA must be executed before this statement has any effect.

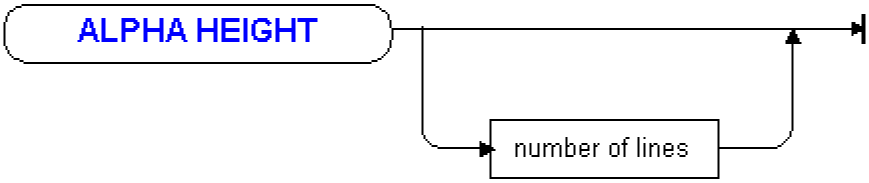
See Also:

CLEAR SCREEN, GRAPHICS, MERGE ALPHA WITH GRAPHICS, PLOTTER IS, SEPARATE ALPHA FROM GRAPHICS

ALPHA HEIGHT

Sets the number of lines used for the ALPHA screen.

Syntax: ALPHA HEIGHT [number-of-lines]



Sample: `ALPHA HEIGHT Num`
`ALPHA HEIGHT 12`

View Sample: **ALPHA HEIGHT.BAS** **(also found in examples directory)**

Description:

The optional number-of-lines is a numeric expression rounded to an integer and must be nine or greater. The bottom number-of-lines of the CRT are reserved for the alpha display. This can be useful in reserving the top of the CRT for the display of graphics. This command is equivalent to a CONTROLCRT,13;lines. If the number-of-lines is not specified, it is reset to the default.

If you are using SEPARATE ALPHA FROM GRAPHICS, you must specify when and where a pen-number that intersects with the alpha write enable mask. For example, on a 256 color display, the mask is 192 when SEPARATE. If pen-numbers of 0 to 7 are used, they won't intersect the mask and no alpha text will be written.

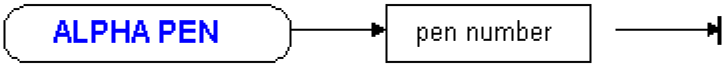
See Also:

ALPHA PEN, KBD LINE PEN, KEY LABELS PEN, PRINT PEN

ALPHA PEN

Sets the ALPHA display color.

Syntax: ALPHA PEN pen-number



Sample:

```
ALPHA PEN Color
ALPHA PEN 137
IF Red THEN ALPHA PEN 2
```

View Sample: **ALPHA PEN.BAS** **(also found in examples directory)**

Description:

This statement overrides any **ALPHA PEN**, PRINT PEN, KBD LINE PEN or KEY LABELS PEN statements in effect. The pen-number is a numeric expression rounded to an integer. Legal values are from 0 to 255. This statement is equivalent to CONTROL CRT,5;pen-number.

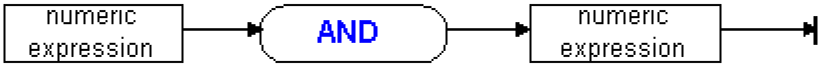
See Also:

COLOR, KBD LINE PEN, KEY LABELS PEN, PRINT PEN

AND

Performs the logical conjunction of two expressions.

Syntax: numeric-expression AND numeric-expression



Sample:
```
IF A AND B THEN C
First=Last AND Ready
A=Age>19 AND Reply$="YES"
```

View Sample: **AND.BAS** **(also found in examples directory)**

Description:

AND returns a value of one (true) or zero (false) from the logical conjunction of two expressions. The value of j **AND** k, where j and k are themselves numeric expressions is one (true) only if both j and k are non-zero. It is zero (false) if either or both j and k are zero. **AND** can be used in combination with other logical or math operators in numeric expressions.

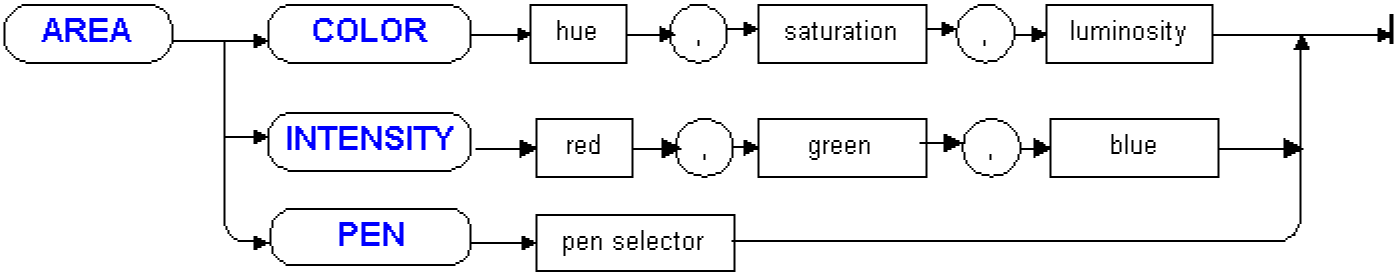
See Also:

EXOR, OR, NOT

AREA

Sets or defines an AREA fill color.

Syntax: AREA COLOR hue, saturation, luminosity
AREA INTENSITY red, green, blue
AREA PEN pen-number



Sample:

```
AREA COLOR Hue,Sat,Lum
AREA INTENSITY Red(I),Green(I),Blue(I)
AREA PEN 11
AREA PEN —Numb
```

Description:

AREA allows you to specify the color used to fill areas. See COLOR for an explanation of how to specify colors with **COLOR, INTENSITY** and **PEN**. The effect of different pen numbers is given in the Drawing Mode Table, below.

If you specify a color with **COLOR** or **INTENSITY** which cannot be produced on the computer system you are using, the color may be approximated by using an available color which is close to the color specified. On some displays this may include dithering available colors to produce a color closer to the one you specified. If dithering is used, the statement will execute slower than an **AREA PEN** statement.

The default area fill color is PEN one. The color defined by **AREA** remains the area fill color until an **AREA**, GINIT or SCRATCH A is executed. IPLOT, PLOT, RPLOT or SYMBOL can also be used to change the area fill color.

Drawing Mode Table

The writing mode of the pen is specified by the current drawing mode and the sign of the pen number. GESCAPECRT,4 is used to change to normal drawing mode. GESCAPECRT,5 is used to change to alternate drawing mode. The following table defines the different writing modes available. P is a positive pen number, X is the present value of a pixel.

Statement	**GESCAPE CRT,4** **Normal**	**GESCAPE CRT,5** **Alternate**
AREA PEN P	P	BINIOR(X,P)
AREA PEN 0	0	0
AREA PEN -P	BINAND(X,BINCMP(P))	BINAND(X,BINCMP(P))

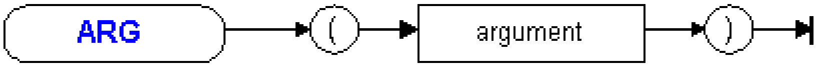
See Also:

COLOR, GESCAPE, IPLOT, PEN, PLOT, RPLOT, SYMBOL

ARG

Returns the Argument (Angle) of a complex number.

Syntax: ARG(numeric-expression)



Sample: `PRINT "Angle = ";ARG(CMPLX(1,2))`

View Sample: **ARG.BAS** **(also found in examples directory)**

Description:

The Argument of a complex number is the angle in the complex plane between the positive real axis and a vector to the complex number. Positive angles are counter-clockwise from the positive real axis. **ARG** returns the principal value which has a range of -PI to PI radians or -180 to 180 degrees, depending on the current trigonometric mode. Note that the **ARG** of a real number can be either 0 or PI (180), depending on whether the number is positive or negative. COMPLEX numbers are stored in rectangular form, but may be used in polar form using ABS and **ARG**. For example:

```
PRINT "Magnitude = ";ABS(Z),"Angle = ";ARG(Z)
```

To enter a number in polar form, convert it from polar form to rectangular in this manner:

```
10  INPUT Magnitude,Angle
20  Z=CMPLX( Magnitude*COS(Angle), Magnitude*SIN(Angle))
```

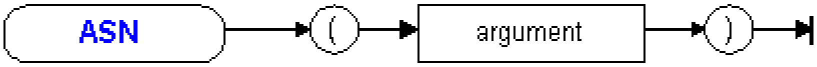
See Also:

ABS, CMPLX, DEG, IMAG, RAD, REAL

ASN

Returns the arcsine of an expression.

Syntax: ASN(numeric-expression)



Sample:

```
Beta=ASN(T1)
PRINT "Angle = ";ASN(Sine)
```

View Sample: **ASN.BAS** **(also found in examples directory)**

Description:

The arcsine of a number is the angle whose sine is that number. **ASN** returns a value between ±PI/2 radians or ±90 degrees. The default trigonometric mode is radians unless changed with the DEG statement. Its argument must be a value between -1 and 1 inclusive.

COMPLEX Arguments

ASN accepts either a COMPLEX or REAL argument and returns a value of the same type. For COMPLEX arguments the angle is returned in radians, regardless of the current trigonometric mode. **ASN** returns the principal value, defined (in terms of complex arithmetic) as

ACS(Z) = CMPLX(0,-1)*LOG(CMPLX(0,1)*Z+SQRT(1-Z^2))

which returns a real part in the range -PI/2 to PI/2. The domain for COMPLEX arguments includes all points in the complex plane (but for REAL arguments, the domain is still -1 to 1, inclusive). Notice that intermediate values generated during the calculation of the function can cause over or underflow errors for very large or small values of Z.

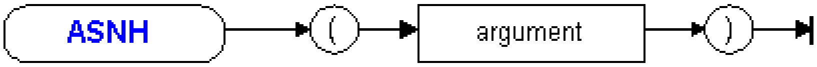
See Also:

ACS, ATN, COS, SIN, TAN, ASNH, ACSH, ATNH, COSH, SINH, TANH, DEG, PI, RAD

ASNH

Returns the hyperbolic arcsine of an expression.

Syntax: ASNH(numeric-expression)



Sample:
```
Beta=ASNH(T1)
PRINT "Angle = ";ASNH(Z)
```

View Sample: **ASNH.BAS** **(also found in examples directory)**

Description:

The hyperbolic arcsine of a number is the angle whose hyperbolic sine is that number. The angle is returned in radians, regardless of the current trigonometric mode. **ASNH** returns the principal value, defined (in terms of complex arithmetic) as

ASNH(Z) = LOG(Z+SQRT(Z^2+1))

which returns an imaginary part in the range -PI/2 to +PI/2. **ASNH** accepts either a COMPLEX or REAL argument and returns a value of the same type. **ASNH** is defined at all points for both COMPLEX and REAL arguments. However, intermediate values generated during the calculation of the function can cause over or underflow errors for very large or small values of Z.

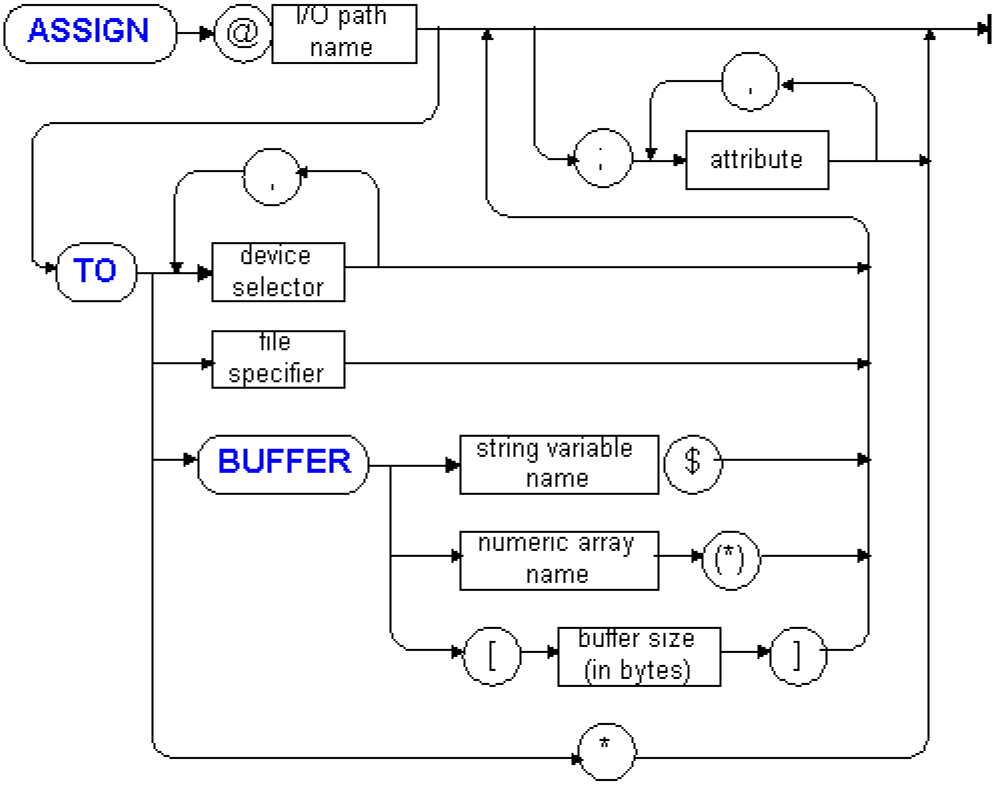
See Also:

ACSH, ASN, ATNH, COSH, SINH, TANH

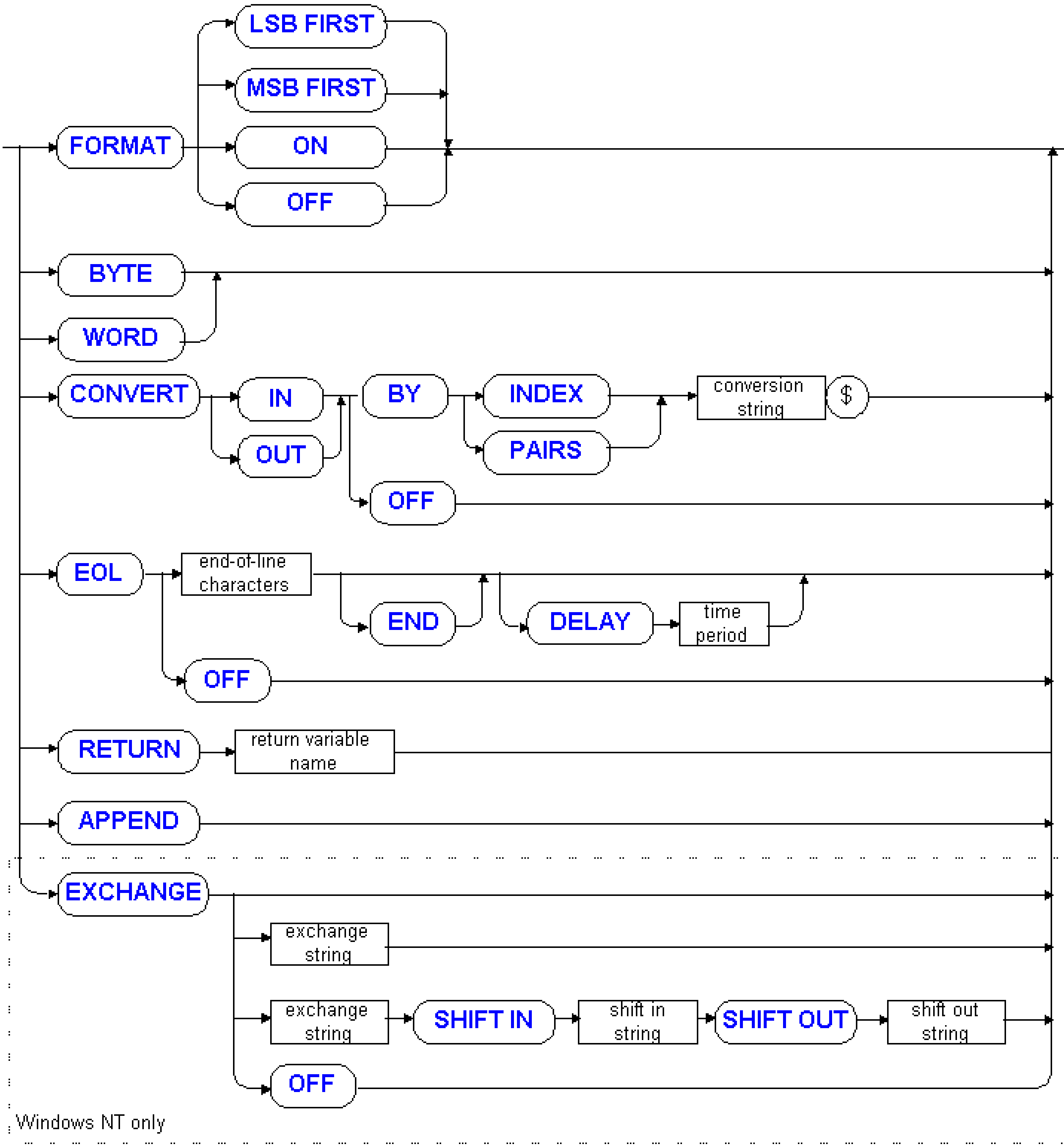
ASSIGN

Sets up an I/O path and its attributes, or creates/destroys widgets.

Syntax: ASSIGN @io-path [TO resource] [;attrib [,attrib...]]
ASSIGN @io-path TO *



literal form of attributes:



View Sample: ASSIGN.BAS **(also found in examples directory)**

where: resource = device-selector [,device-selector...] |
file-specifier |

pipe-specifier |
BUFFER {string-name$ | numeric-array(*) | [buf-size]}
attrib = FORMAT {ON|OFF|MSB FIRST|LSB FIRST} | {BYTE | WORD} |
CONVERT {IN|OUT} {OFF | {BY {INDEX|PAIRS} convert$}} |
PARITY {EVEN | ODD | ONE | ZERO | OFF} |
EOL eol-chars [END] [DELAY seconds] | EOL OFF |
RETURN numeric-name | APPEND
buf-size = size of the buffer in bytes
convert$ = string-name. If INDEX, it can have up to
256 characters. If PAIRS, it must have an even
number of characters.
eol-chars = string-expression of up to 8 characters
seconds = numeric-expression rounded to the nearest
0.001 through 32.767 (default is 0)

Sample:

```
ASSIGN @Code TO Isc;FORMAT OFF
ASSIGN @Close TO *
ASSIGN @Devices TO 711,712,715
ASSIGN @Buf1 TO BUFFER Str1$
ASSIGN @B TO BUFFER [12800]
ASSIGN @Buffer TO BUFFER Array(*)
ASSIGN @File TO "C:\MSDOS\FILE2"
ASSIGN @File TO "/unix/CityDir/StFile";APPEND
ASSIGN @T TO 12;WORD,RETURN R,EOL My$ DELAY 1
ASSIGN @Stdout TO "| cat";EOL CHR$(10)
ASSIGN @Pipe TO "finger |"
ASSIGN @Panel TO WIDGET “PANEL”;SET(“X”:5,”Y”:5,
“WIDTH”:500,“HEIGHT”:350,“TITLE”:”Engine Monitor”)
ASSIGN @Strip TO WIDGET “STRIPCHART”;PARENT@Main1,SET
(“X”:5,”Y”:5,”WIDTH”:350,”HEIGHT”:250,”SHOW NUMBERING”:0)
ASSIGN @strip TO * !Destroy the @Strip widget
```

Description:

ASSIGN makes a connection between a file, buffer, device, or devices and an I/O path name. An I/O path contains the necessary information to control the input or output of data. It is used in I/O statements to specify the source or destination of the input or output. An I/O path name can be placed in a COM statement and can be passed by reference as an argument to subprograms. I/O operations can be re-directed by re-**ASSIGN**ing the I/O path. **ASSIGN** may also be used to change previous I/O path attributes or to close an I/O path.

The ASSIGN statement can be used to:

- Create a new level-0 widget
- Create a widget as a child of an existing widget
- Create a transient widget
- Destroy an existing widget

Within the **ASSIGN** statement, a “widget handle” (equivalent to an I/O path) is associated with the new widget. The widget handle can be used in subsequent statements, such as STATUS, CONTROL, and ON EVENT, to control the appearance and behavior of the widget.

Also, the *widget handle* names the widget to be destroyed when ASSIGN@*widget handle* TO * is used to destroy a widget.

Devices

To do I/O with an IEEE-488 device which has a primary address of 2, you would use the **ASSIGN** statement (assuming the default IEEE-488 interface select code of 7):

```
ASSIGN @io-path TO 702
```

To do I/O with a device hooked to the serial port (assuming the port is at the default ISC of 9), you would use:

```
ASSIGN @io-path TO 9
```

A device can have more than one I/O path name (each with different attributes) associated with it.

An I/O path name can have more than one device assigned to it. If multiple devices are specified, they must be on the same interface. When OUTPUT is made to an I/O path assigned to multiple devices, all the devices receive the data. When ENTER is made from multiple devices, the first device specified sends data to the computer and to all the other devices assigned to the I/O path name. When CLEAR, LOCAL, PPOLL CONFIGURE, PPOLL UNCONFIGURE, REMOTE or TRIGGER are made on multiple devices, all the devices receive the IEEE-488 message.

Files

A file is opened when the **ASSIGN** statement specifies a file-specifier. The file's position pointer is set to the beginning of the file unless **APPEND** is specified and is updated to point to the next byte to be read or written with each ENTER or OUTPUT statement.

Buffers

The statement

```
ASSIGN @Io_path TO BUFFER [300]
```

creates an unnamed buffer and assigns it a named I/O path. The

```
ASSIGN @Io_path TO BUFFER X(*)
```

statement assigns an I/O path name to a buffer variable previously declared in a COM, COMPLEX, DIM, INTEGER or REAL statement. The buffer specified in **ASSIGN** may now be used in ENTER, OUTPUT or TRANSFER statements. Buffer control information can be read with the STATUS statement and includes the current number of bytes in the buffer (initially set to 0), the empty and the fill pointers (initially set to 1) and the buffer capacity.

An I/O path name must exist for as long as its assigned buffer exists. To insure this, the following rules are used: Buffers cannot be declared in ALLOCATE statements. For a named buffer and its associated I/O path name, if either appear in a COM block, then the other must also. The same is true of subprogram parameters or else the buffer must appear in a COM block accessible to the subprogram. I/O path names assigned to unnamed buffers cannot appear in COM blocks or subprogram parameters.

Unnamed buffers can only be accessed through their I/O path names. When the I/O path of an unnamed buffer is closed, the buffer space is deallocated. Named buffers can be directly accessed through their variable names, although this is not generally recommended. It does not perform necessary byte order swapping. And the data in the buffer can be changed without proper update of the buffer control registers (empty and fill pointers, current number of bytes). To automatically update the buffer control registers use the ENTER, OUTPUT, and TRANSFER statements.

Binary data in a buffer exists in the byte order of the data source. If that order is different than the byte order of the computer, then accessing the data through the variable name results in incorrect data. Again, using ENTER, OUTPUT and TRANSFER to access the data handles the byte order correctly.

FORMAT

The **FORMAT** option controls whether data is handled in binary or ASCII. If **FORMAT** is not explicitly specified a default format is used as specified in the following. In addition to the HP BASIC compatible **FORMAT ON** and **FORMAT OFF** options, HTBasic also allows the **FORMAT MSB FIRST** and **FORMAT LSB FIRST** options. These options allow explicit specification of the data byte ordering. If **LSB FIRST** is specified, then numbers are sent and received with the Least Significant Byte first. If **MSB FIRST** is specified, then numbers are sent and received with the Most Significant Byte first.

LSB is the native byte order for HTBasic. If a device is capable of sending binary data in **LSB** format, it should be instructed to do so and **FORMAT LSB FIRST** should be specified instead of **FORMAT OFF**.

BYTE and WORD

When **BYTE** is included in the **ASSIGN** statement the data is sent and received as 8-bit bytes. **WORD** sends and receives data in 16-bit words and can only be used on a 16-bit interface. The default form if neither **BYTE** nor **WORD** is explicitly specified is **BYTE**.

CONVERT

When **CONVERT** is included in the **ASSIGN** statement a character-conversion table is used during OUTPUT and ENTER operations (**OUT** converts during OUTPUT and **IN** converts during ENTER). The default attribute is no conversion (**CONVERT IN OFF** and **CONVERT OUT OFF**). If **CONVERT OUT** is specified then conversions are made after EOL characters are appended but before parity generation (if **PARITY** specified). If **CONVERT IN** is specified then conversions are made after parity check but before item or statement terminators are checked.

Note: CONVERT is not supported in HTBasic.

When **BY INDEX** is included, an index system is used in the conversion process. Each original character is used as an index into the conversion string. CHR$(1) is replaced by the 1st character, CHR$, (2) is replaced the 2nd character, etc. Note however that CHR$, (0) is replaced by the 256th character in the conversion string.

When **BY PAIRS** is included, pairs of characters are used in the conversion process (the original character and its replacement character). The original characters (odd characters) are searched in the conversion string. If the original is found it is replaced by the next (replacement) character. If the original is not found, then no conversion takes place.

PARITY

The most significant bit of the byte is considered the parity bit. On OUTPUT, parity is calculated after any **CONVERT**. On ENTER, parity is checked before any **CONVERT**.

Note: The **PARITY** option to **ASSIGN** is not supported in HTBasic. The parity for the serial interface should be set using the appropriate CONTROL register.

EOL

The default End-Of-Line is a carriage-return (CR) and line-feed (LF) sent with no END indication and no **DELAY**. Specifying END causes an interface specific END indication to be sent with the EOL. On the IEEE-488, END causes EOI to be sent with the final character of the EOL. Specifying **DELAY** causes the computer to pause for the specified number of seconds after sending the EOL and before allowing the program to continue. The delay time depends on the timing resolution available on the computer you are using. The default EOL can be restored by specifying **EOL OFF**.

Note: LF or CR/LF are always used to terminate ENTER data, regardless of the setting of **EOL** in the **ASSIGN** statement.

RETURN

RETURN can be used with **ASSIGN** to test whether the **ASSIGN** operation was successful. If not successful the error number is returned in the variable specified, otherwise a zero is returned.

APPEND

If **APPEND** is specified, the file position is moved to the end-of-file after the **ASSIGN**. If it is not specified, the file position is moved to the beginning of the file. **APPEND** is supported on BDAT and ordinary files, but not LIF ASCII files.

Close I/O Paths

Closing an I/O path makes the path invalid. All subsequent ON event statements for the closed I/O path are not acted upon. If an I/O path name has not been declared in a COM statement it may be closed in the following ways:

1. Explicitly close a path by executing: **ASSIGN** @io-path TO *
2. Re-assigning the I/O path: **ASSIGN** @path **TO** resource
3. Exiting the subprogram: SUBEND, SUBEXIT, ON...RECOVER, or RETURN...
4. Stopping the program: END, GET, LOAD, SCRATCH, SCRATCH A, SCRATCH C or STOP

If an I/O path name has been declared in a COM statement it may be closed in the following ways:

1. Explicitly close a path by executing: **ASSIGN** @io-path **TO** *
2. Executing SCRATCH A or SCRATCH C
3. Executing EDIT, GET, LOAD in a program that has a COM statement that does not match the COM statement that contains the I/O path name.

Changing Attributes

The attributes of a previously **ASSIGN**ed I/O path may be individually changed by omitting "**TO** resource" in the **ASSIGN**statement. To restore all default attributes use ASSIGN@io-path.

PARENT Option

If no parent is specified when creating a new-widget, the widget is said to be a “level-0” widget. A level-0 (Zero) widget is not constrained to be within another widget, and may exist at any place in the HTBasic for Windows output window. The X and Y coordinates of the widget are relative to the upper-left corner of the HTBasic for Windows output window.

Only level-0 widgets may include a title bar, a resize border, and a system menu. The title bar and resize border allow you to change the position and size of the widget. If a parent is specified, the new widget will be treated as a “child widget” of its parent. If you attempt to move a child widget outside the border of the parent widget, the child will be “clipped” at the parent widget’s borders. The child widget’s X and Y coordinates are relative to the upper-left corner of the parent widget.

Not all widgets can be parents, and not all widgets can be children of parent widgets.

TRANSIENT Option

The TRANSIENT option is used primarily when the resulting widget is to function as a dialog. If you create a widget using the TRANSIENT option, other non-transient widgets cannot be placed on top of the widget.

If the transient widget has a parent, the transient widget is not restricted to lie within the bounds of its parent as are other child widgets. Visually, the transient widget appears to be a special type of level-0 widget.

SET Option

All widgets have a variety of attributes that control their appearance and behavior. You can initialize the values of these attributes at the time of creation of the widget by using the SET option.

Attributes are either scalar (may contain a singel value) of vector (may be assigned an array of values) and have value of either numeric or string type.

Shorthand: Assigning Attributes

You can use a shorthand method to assign values to several scalar attributes without naming them individually on the **ASSIGN** statement.   To do this, you store all the attributes in a string array and all the matching values in another array of the same size.

Then, when you specify both array names in the SET option of the **ASSIGN** statement, the attribute named in each element of the string array will be assigned the corresponding value in the value array. Elements of the string array that contain nothing, or nothing but blanks, will be ignored.

For example:

```
Attribs$(1) = “X”
Attribs$(2) = “Y”
Attribs$(3) = “WIDTH”
Attribs$(4) = “HEIGHT”
Values(1) = 5
Values(2) = 5
Values(3) = 500
Values(4) = 300
```

ASSIGN @Panel TO WIDGET “PANEL”;SET(Attrib$(*):Values(*))

Porting From HP BASIC:

When an **ASSIGN** fails, the previous state of the I/O path is not preserved. Also, the **CONVERT** and **PARITY** options are not implemented.

If changes are made to an **ASSIGN**ed file, the directory entry is not updated until the file is closed. The Operating System buffers reads and writes to disk. You should not remove a diskette or turn the power off while a file is **ASSIGN**ed. Exchanging diskettes while a file is **ASSIGN**ed on the first can destroy the next diskette. Two I/O paths **ASSIGN**ed simultaneously to the same file can produce slightly different results than HP BASIC, depending on the buffering the OS does.

The HTBasic **ASSIGN** includes two new options, **FORMAT LSB FIRST** and **FORMAT MSB FIRST**, to specify byte ordering of binary numeric data transfers. This provides the ability to do binary transfers with any device or computer, regardless of the byte ordering that device uses.

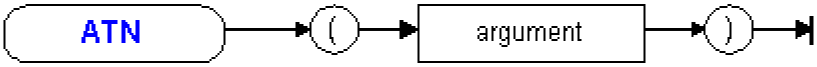
See Also:

CREATE, CREATE ASCII, CREATE BDAT, PURGE, ENTER, OUTPUT

ATN

Returns the arctangent of an expression.

Syntax: ATN(numeric-expression)



Sample:

```
C2=ATN(4.5)
PRINT "Angle = ";ATN(Ang1)
```

View Sample: **ATN.BAS** **(also found in examples directory)**

Description:

The arctangent of a number is the angle whose tangent is that number. ATN returns a value between ±PI/2 radians or ±90 degrees, depending on the current trigonometric mode. The default trigonometric mode is RAD. Use DEG to change to degrees.

COMPLEX Arguments

ATN accepts either a COMPLEX or REAL argument and returns a value of the same type. For COMPLEX arguments the angle is returned in radians, regardless of the current trigonometric mode. ATN returns the principal value, defined (in terms of complex arithmetic) as

ATN(Z) = CMPLX(0,1/2)*LOG((CMPLX(0,1)+Z)/(CMPLX(0,1)-Z))

which returns a real part in the range -PI/2 to PI/2. The domain for COMPLEX arguments includes all points in the complex plane except CMPLX(0,1). Notice that intermediate values generated during the calculation of the function can cause over or underflow errors for very large or small values of Z.

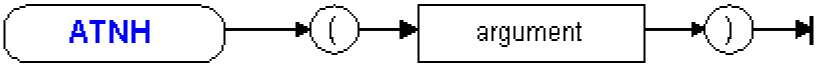
See Also:

ACS, ASN, COS, SIN, TAN, ASNH, ACSH, ATNH, COSH, SINH, TANH, DEG, PI, RAD

ATNH

Returns the hyperbolic arctangent of an expression.

Syntax: ATNH(numeric-expression)



Sample:

```
C2=ATNH(CMPLX(4.5,2))
PRINT "Angle = ";ATNH(Z)
```

View Sample: **ATNH.BAS** **(also found in examples directory)**

Description:

The hyperbolic arctangent of a number is the angle whose hyperbolic tangent is that number. The angle is returned in radians, regardless of the current trigonometric mode. **ATNH** accepts either a COMPLEX or REAL argument and returns a value of the same type. For REAL arguments the domain is between -1 and 1. For complex arguments, **ATNH** returns the principal value, defined (in terms of complex arithmetic) as

ATNH(Z) = 1/2*LOG((1+Z)/(1-Z))

which returns an imaginary part in the range -PI/2 to PI/2. The domain for COMPLEX arguments includes all points in the complex plane except CMPLX(±1,0). Notice that intermediate values generated during the calculation of the function can cause over or underflow errors for very large or small values of Z.

See Also:

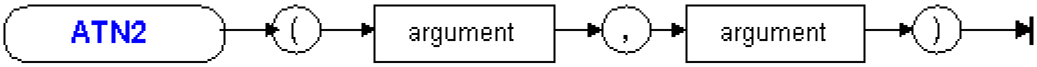
ACSH, ASNH, COSH, SINH, TANH

ATN2

Returns the angle to a point.

Syntax: ATN2(y, x)

where: x and y = numeric-expressions



Sample: `PRINT "Angle=";ATN2(1,2)`

View Sample: **ATN2.BAS (also found in examples directory)**

Description:

ATN2(x,y) returns the angle between the positive real x-axis and a vector to the point (x,y). Positive angles are counter-clockwise from the x axis. **ATN2** returns a value in the range of -PI to PI radians or -180 to 180 degrees, depending on the current trigonometric mode. **ATN2**(0,0) is undefined and causes an error.

ATN2(y,x) is so named because of its similarity to ATN(y/x). However, ATN(y/x) does not calculate correct angles for points in the 2nd and 3rd quadrants. In some languages, this function is named ANGLE(x,y). In HTBasic, it is named **ATN2**(y,x) to match HP Series 80 BASIC.

Porting to HP BASIC:

ATN2 is a new HTBasic function that is not available in HP BASIC. It should not be used in programs that must be ported back to HP BASIC.

See Also:

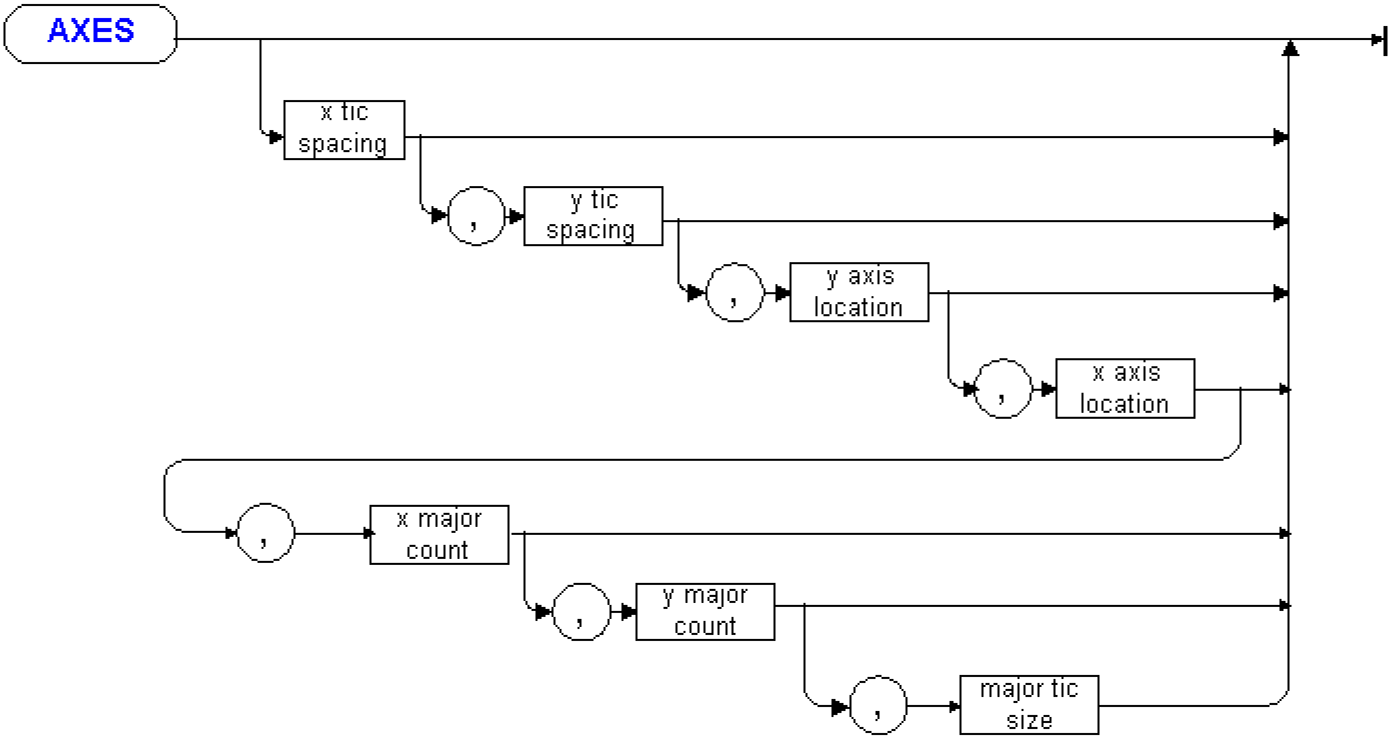
ABS, ARG, DEG, RAD, SQRT

AXES

Draws x-y axes.

Syntax: AXES [x1 [,y1 [,x2 [,y2 [,x3 [,y3 [,major]]]]]]]

where: x1,y1 = numeric-expressions, x,y tick spacing
x2,y2 = numeric-expressions, x,y origin of axis
x3,y3 = numeric-expressions, rounded to integers, major tick counts
(range 1 through 32767)
major = numeric-expression, rounded to an integer, major tick size



Sample: `AXES 5,5,0,100`
`AXES X,Y,Midx,Midy,Maxx/10,Maxy/10`

View Sample: **AXES.BAS** **(also found in examples directory)**

Description:

The **AXES** statement draws X-Y axes. You may specify the tic spacing on each axis in WINDOW units by giving two arguments, one for the x tic spacing and one for the y tic spacing; the default 0,0 means don't draw ticks. You may then specify the axes origin in WINDOW units; the default is 0,0. Also, you may specify the number of ticks between major tick marks; the default is 1,1 meaning that every tick is major. Lastly, you may specify the major tick size in VIEWPORT units; the default is 2.

The axes extend across the soft-clip area and the tick marks are symmetric about the axes but are clipped by the soft-clip area. If the x or y axis is outside the clip area, then tick marks are drawn into the non-clip area. The axes and tick marks are drawn in the current line style and pen color. A major tick is placed at the axis origin. The minor tick marks are half the size of the major tick marks.

See Also:

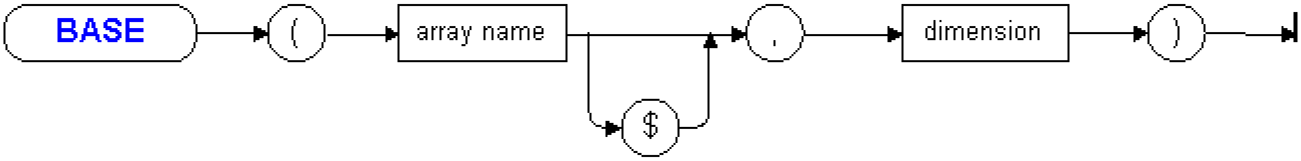
FRAME, GRID, LINE TYPE, PEN

BASE

Returns the lower bound of an array dimension.

Syntax: BASE(array-name[$],dimension)

where: dimension = integer between 1 and 6 ≤ RANK of array



Sample:

```
Lwr=BASE(Yarray$,Dim)
Uppr(2)=BASE(A,2)+SIZE(A,2)-2
```

View Sample: **BASE.BAS** **(also found in examples directory)**

Description:

BASE returns the current lower bound of an array dimension. This might be different than the DIMensioned value if a REDIM or matrix statement has changed it. This function is also useful in a subprogram where an array is passed in as one of the parameters.

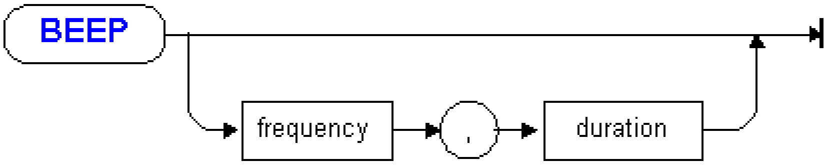
See Also:

ALLOCATE, DIM, OPTION BASE, RANK, REDIM, SIZE

BEEP

Generates music or sound effects.

Syntax: BEEP [frequency, duration]



Sample:

```
BEEP
BEEP Tone,Seconds
BEEP Freq,Duration
BEEP 75.5*Freq,Sec
```

View Sample: **BEEP.BAS** **(also found in examples directory)**

Description:

BEEP generates a frequency for a specified duration in seconds. On computers that do not provide control for variable frequency sound generation, **BEEP** generates a beep or bell sound. The range of the duration is 0 to 2.55 and is rounded to the nearest 0.01 seconds, subject to the timing resolution of your computer system. The value 2.55 is used for any duration greater than 2.55. If no frequency or duration is specified, a 1220.7 Hz beep is generated for 0.2 seconds.

Usage Notes

The period (not the frequency) is rounded to a multiple of 0.838 micro-seconds. The range of frequencies is 40.7 Hz to 32.767 KHz. (HP BASIC rounds the frequency value to a multiple of 81.38 Hz and supports a range of 81 Hz to 5.208 KHz.)

See Also:

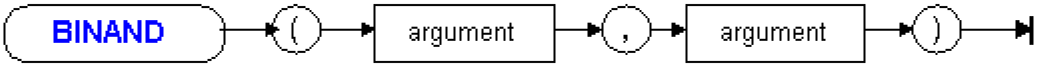
SOUND

BINAND

Performs a bit by bit logical AND.

Syntax: BINAND(arg, arg)

where: arg = numeric-expression rounded to an INTEGER range -32768 to +32767



Sample:

```
I=BINAND(J,K)*6
IF BINAND(Low,4) THEN CALL Set
```

View Sample: **BINAND.BAS** **(also found in examples directory)**

Description:

Use **BINAND** to clear or test specific bits. **BINAND**(A,B) converts the values of A and B to integers. The integer values of A and B are then treated as unsigned binary numbers. Corresponding bits in A and B are then ANDed together. If both corresponding bits in A and B are a 1 the resulting bit is set to a 1 otherwise it is set to a 0. The following example:

```
BINAND(12,6)
```

performs a bit by bit logical AND of 12 with 6.

12	= 0000000000001100
6	= 0000000000000110
BINAND(12,6)	= 0000000000000100

The resulting binary number represents 4.

See Also:

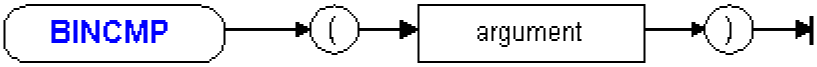
BINCMP, BINEOR, BINEQV, BINIMP, BINIOR, BIT, ROTATE, SHIFT

BINCMP

Performs a bit by bit complement.

Syntax: BINCMP(arg)

where: arg = numeric-expression rounded to an INTEGER



Sample:
```
B=BINCMP(A)
```

View Sample: **BINCMP.BAS** **(also found in examples directory)**

Description:

The result of **BINCMP**(A) is calculated by first converting the value of A to an integer. The integer value of A is then treated as a binary number. Each bit of the result is set to 1 if the corresponding bit of A is 0 and is set to 0 if the corresponding bit of A is 1. Here is an example of how **BINCMP** works:

```
BINCMP(13)
```

The number 13 is considered a binary number, then the bitwise complement is performed:

13	= 0000000000001101
BINCMP(13)	= 1111111111110010

The resulting binary number represents -14.

See Also:

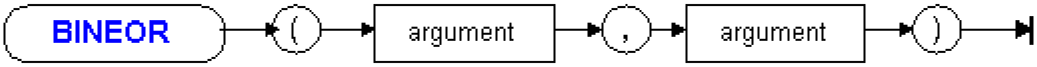
BINAND, BINEOR, BINEQV, BINIMP, BINIOR, BIT, ROTATE, SHIFT

BINEOR

Performs a bit by bit exclusive OR (EXOR).

Syntax: BINEOR(arg, arg)

where: arg = numeric-expression rounded to an INTEGER



Sample:

```
M=BINEOR(J,K)
Toggle=BINEOR(Toggle,4)
```

View Sample: **BINEOR.BAS** **(also found in examples directory)**

Description:

BINEOR is useful when you want to "toggle" a certain bit or bits. **BINEOR**(A,B) converts the values of A and B to integers. The integer values of A and B are then treated as unsigned binary numbers. Each bit of the result is set to 1 if exactly one of the corresponding bits for either A or B is 1 and is set to 0 if the corresponding bits of A and B are both 0 or both 1. An example of **BINEOR** follows:

```
BINEOR(12,6)
```

The numbers 12 and 6 are considered binary numbers, then the bitwise exclusive OR is performed.

12	= 0000000000001100
6	= 0000000000000110
BINEOR(12,6)	= 0000000000001010

The resulting binary number represents 10.

See Also:

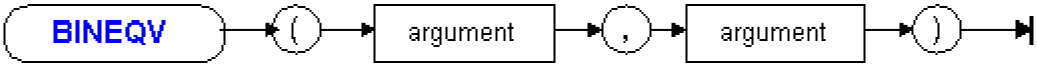
BINAND, BINCMP, BINEQV, BINIMP, BINIOR, BIT, ROTATE, SHIFT

BINEQV

Performs a bit by bit equivalence operation.

Syntax: BINEQV(arg, arg)

where: arg = numeric-expression rounded to an INTEGER



Sample:

```
J=BINEQV(&HFF00,Var)
I=BINEQV(15,J)
```

View Sample: **BINEQV.BAS** **(also found in examples directory)**

Description:

The result of **BINEQV**(A,B) is calculated by converting A and B to integer values. Then each bit of the result is set to 1 if the corresponding bits in A and B are equal. This table illustrates this relationship.

A	B	BINEQV(A,B)
0	0	1
0	1	0
1	0	0
1	1	1

The following example:

```
BINEQV(12,6)
```

performs a bit by bit equivalence of 12 and 6.

12	= 0000000000001100
6	= 0000000000000110
BINEQV(12,6)	= 1111111111110101

The resulting binary number represents -11.

Porting to HP BASIC:

BINEQV is a new HTBasic function that is not available in HP BASIC. It should not be used in programs that must be ported back to HP BASIC.

See Also:

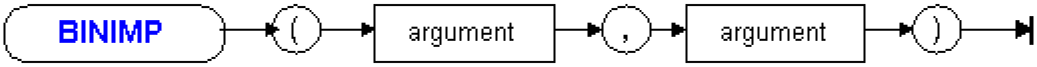
BINAND, BINCMP, BINEOR, BINIMP, BINIOR, BIT, ROTATE, SHIFT

BINIMP

Performs a bit by bit implication operation.

Syntax: BINIMP(arg, arg)

where: arg = numeric-expression rounded to an INTEGER



Sample:

```
K=BINIMP(Var,&O377)
I=BINIMP(12,J)
```

View Sample: **BINIMP.BAS** **(also found in examples directory)**

Description:

The result of **BINIMP**(A,B) is calculated by converting A and B to integer values. Then each bit of the result is set to 1 or 0 depending on the corresponding bits in A and B. The following truth table defines the implication operation:

A	B	BINIMP(A,B)
0	0	1
0	1	1
1	0	0
1	1	1

Note that the operation is not commutative. That is, **BINIMP**(A,B) <> **BINIMP**(B,A). The following example:

```
BINIMP(12,6)
```

performs a bit by bit implication of 12 and 6.

12	= 0000000000001100
6	= 0000000000000110
BINIMP(12,6)	= 1111111111110111

The resulting binary number represents -9.

Porting to HP BASIC:

BINIMP is a new HTBasic function that is not available in HP BASIC. It should not be used in programs that must be ported back to HP BASIC.

See Also:

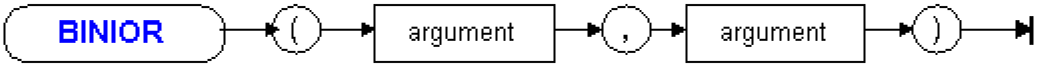
BINAND, BINCMP, BINEOR, BINEQV, BINIOR, BIT, ROTATE, SHIFT

BINIOR

Performs a bit by bit inclusive OR.

Syntax: BINIOR(arg, arg)

where: arg = numeric-expression rounded to an INTEGER



Sample:

```
Set=BINIOR(Byte,Bit)
Msb=BINIOR(-1,2^14)
```

View Sample: **BINIOR.BAS** **(also found in examples directory)**

Description:

BINIOR can be used to set specific bits. **BINIOR**(A,B) converts the values of A and B to integers. The integer values of A and B are then treated as unsigned binary numbers. Each bit of the result is set to 1 if the corresponding bit of either or both A or B is 1, and 0 if the corresponding bits of both A and B are 0. An example of **BINIOR** is:

```
BINIOR(12,6)
```

The numbers 12 and 6 are considered binary numbers, then the bitwise OR is performed.

12	= 0000000000001100
6	= 0000000000000110
BINIOR(12,6)	= 0000000000001110

The resulting binary number represents 14.

See Also:

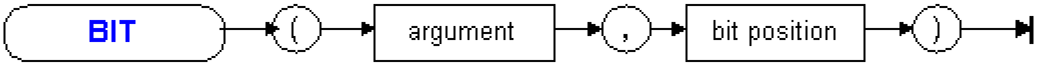
BINAND, BINCMP, BINEQV, BINIMP, BINEOR, BIT, ROTATE, SHIFT

BIT

Allows any bit in an INTEGER to be tested.

Syntax: BIT(arg, bit-position)

where: arg = numeric-expression rounded to an INTEGER
bit-position = numeric-expression rounded to an INTEGER



Sample:

```
Db1=BIT(Db1,4)
Flag=BIT(byte,0)
IF BIT(Byte,Abit) THEN PRINT "Bit #"; Abit;"is on"
```

View Sample: **BIT.BAS** **(also found in examples directory)**

Description:

Use **BIT** to test any bit in an integer without having to manually search the integer for the desired bit value. The bit positions are numbered from 0 to 15 with 0 being the right-most or least significant bit position. If the bit is set **BIT** returns a 1, otherwise **BIT** returns a 0. An example of **BIT** follows:

```
BIT(12,3)
```

The number 12 is considered a binary number and tested in this manner:

12	= 0000000000001100
Bit 3	= 0000000000001000
BIT(12,3)	= 1

The result is 1 because bit 3 is set in the number 12.

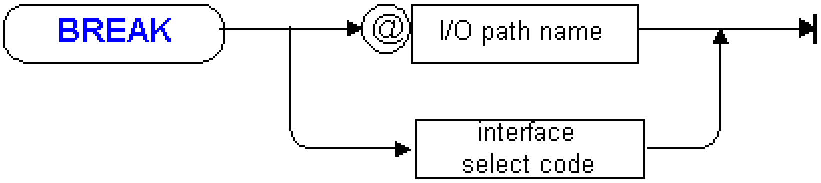
See Also:

BINAND, BINCMP, BINEQV, BINIMP, BINEOR, BINIOR, ROTATE, SHIFT

BREAK

Sends a BREAK on a serial interface.

Syntax: BREAK { @io-path | interface-select-code }



Sample: `BREAK 9`

`BREAK @Serial`

View Sample: **BREAK.BAS** **(also found in examples directory)**

Description:

A **BREAK** signal is sent by manipulating the Data Out signal in the following manner: a logic high of 400-ms is sent followed by a logic low of 60-ms. The **BREAK** is sent immediately. The interface must be a serial interface.

See Also:

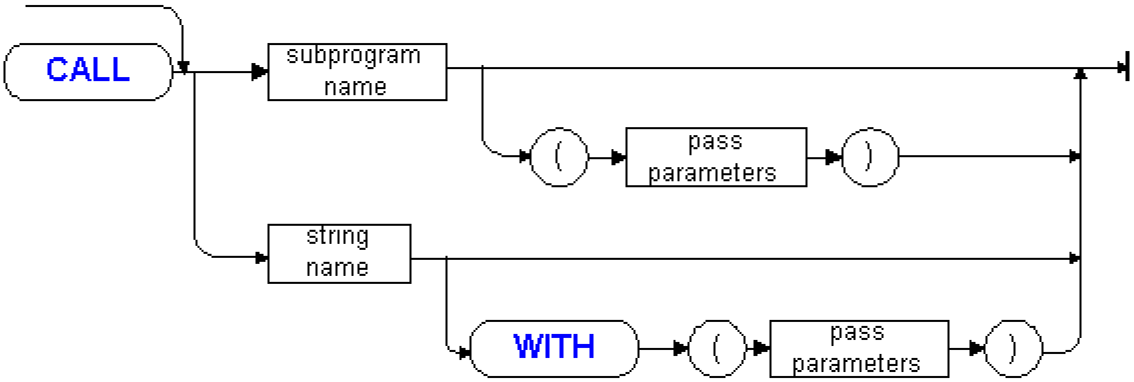
ABORTIO, RESET

CALL

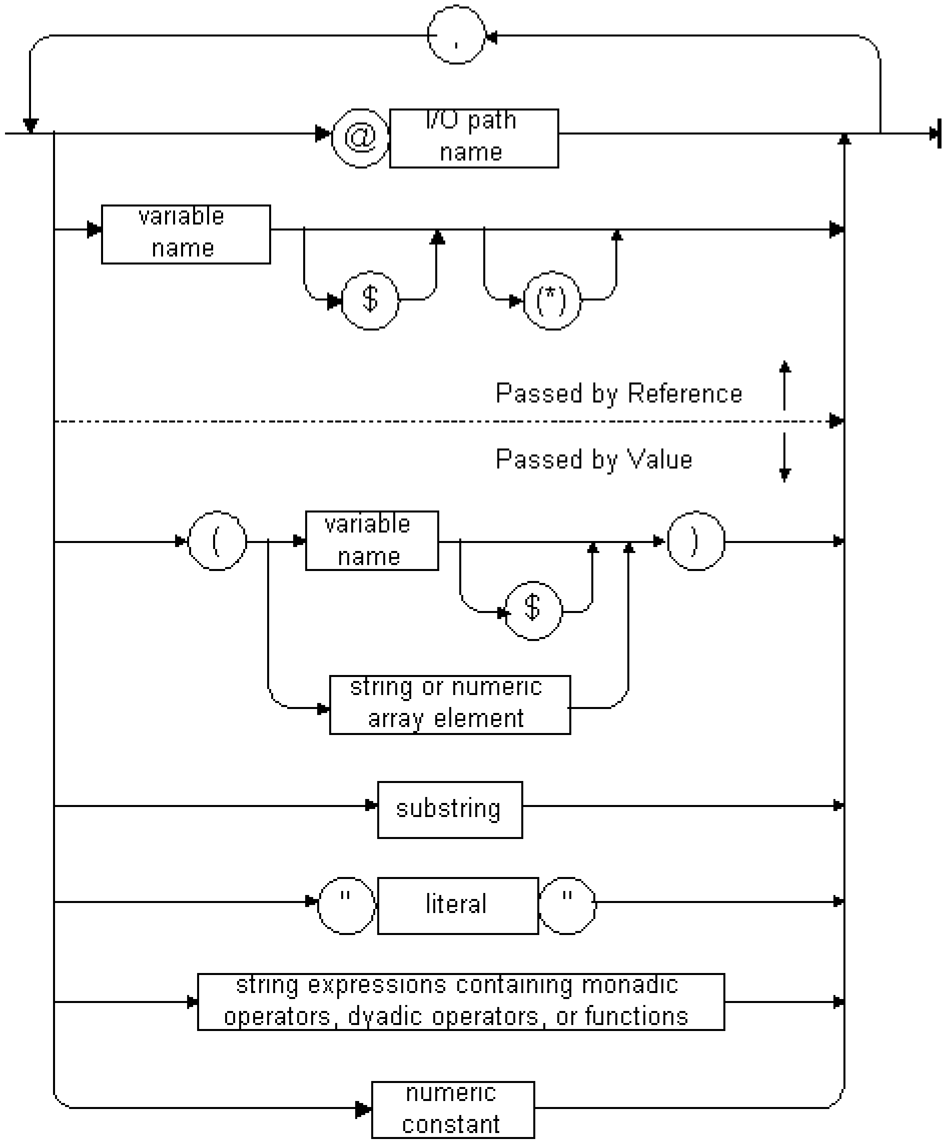
Starts execution of specified SUBprogram or CSUB.

Syntax: [CALL] subprogram-name [(argument [,argument...])]
CALL sub-pointer [WITH (argument [,argument...])]

where: sub-pointer = string expression with subprogram name
argument = pass-by-reference | pass-by-value
pass-by-reference = @io-path | variable-name[$][(*)] |
string-array-element | numeric-array-element
pass-by-value = (variable-name[$]) | numeric-constant |
numeric-expression | (numeric-array-element) |
"string-literal" | string-name$ [(subscripts)] sub-string |
string-expression | (string-array-element)



pass parameters:



Sample: `CALL Deriv(X,Y)`

```
Fft(Array(*))
CALL Test(Ref,(Value),@Source)
CALL A$ WITH (4,1.23,"hello")
```

View Sample: **CALL.BAS** **(also found in examples directory)**

Description:

CALL transfers control to the specified SUBprogram. The context is changed to the SUB and begins running at the statement following the SUB statement. The subprogram continues to run until it encounters a SUBEND or SUBEXIT, at which point control returns to the statement after the **CALL**. If more than one SUB exists with the same name, control is transferred to the SUB with the lowest line number. The name of the SUB may be specified explicitly or in a string expression (sub-pointer):

```
CALL Clayton ! Explicit
CALL "Clay"&"ton" ! String expression
```

CALL may also pass arguments to the subprogram. The list of arguments in the **CALL** statement must match, in type and number, the list of parameters in the SUB statement. The **CALL**

statement may pass the arguments by reference or value as shown in the syntax description above. *Pass-by-value* means that the subprogram receives only the value and cannot change any variables in the calling subprogram. *Pass-by-reference* means that the subprogram is told the variable's location in memory (the variable's address), so that the subprogram can use and modify the variable itself.

The **CALL** keyword may be omitted if the **CALL** statement is alone on a line and the subprogram name is specified explicitly, but if it is part of another statement, such as an IF, then it is required.

Subprogram Pointers

If a string expression specifies the subprogram name in the **CALL** statement, the string expression is called a subprogram pointer because it "points" to the subprogram rather than explicitly naming it. As the expression changes, the pointer points to different subprograms. The following example illustrates how this can be useful.

```
10   SUB Xform(X(*))
20     Method$="Xform"&VAL$(RANK(X))
30     IF NOT INMEM(Method$) THEN LOADSUB Method$
40     CALL Method$ WITH(X(*))
50     DELSUB Method$
60   SUBEND
```

The **CALL** keyword must be used and the subprogram must be specified with the initial character in uppercase and subsequent characters in lowercase. Subprogram pointers can also be used in DELSUB, INMEM, LOADSUB and XREF statements.

Note: If you must write programs portable back to HP BASIC, don't use subprogram pointers in DELSUB, LOADSUB, and XREF statements. Also, HTBasic allows string expressions to be used, while HP BASIC is limited to a simple string variable.

See Also:

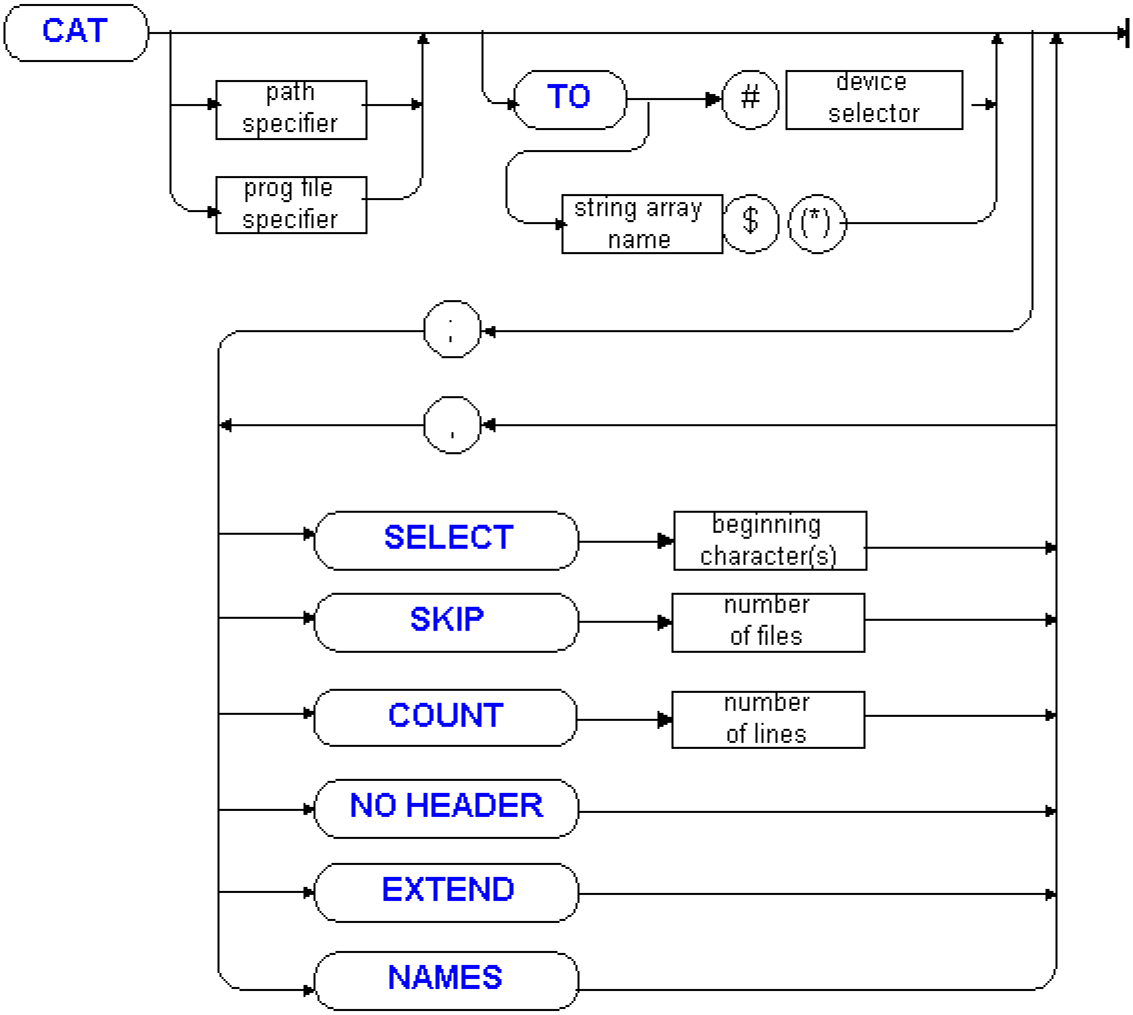
CSUB, DELSUB, LOADSUB, SUB

CAT

Displays a catalog of files or PROG file contexts.

Syntax: CAT [source] [TO destination] [; option [,option...]]

where: source = path-specifier | prog-file-specifier
destination = #device-selector | string-array$(*)
option = COUNT number of lines | EXTEND | NAMES | NO HEADER |
SELECT begin-characters | SKIP number-of-files
begin-characters = string expression



Sample:
```
CAT
CAT "C:\WP";NO HEADER
CAT "A:" TO #701; SELECT "X",SKIP 1;COUNT Count
CAT "*.TXT"
```

View Sample: CAT.BAS **(also found in examples directory)**

Description:

Catalogs of Contexts in a PROG file

If a prog-file-specifier is given, a list of the contexts in that file are listed. The different context types are main context, subprogram contexts, user defined function subprogram contexts and CSUB contexts. Each context is listed with its name, size and type.

Catalogs of Files in a Directory

CAT is used to produce a catalog of files that are present in a directory of a mass storage device. **CAT** can be used as a program command or statement. A header is printed and information is given about each file. The format of the information depends on the file system. However, when **CAT** is directed to a string array, it produces the SRM catalog format regardless of the file type. The **EXTEND** option can be used to suppress the SRM format so that the string array is written with the same format as would be displayed on the screen. The format for each file system, including SRM, is given later in this entry.

If the file name is too long to give in the space provided by each of the following formats, an asterisk, "*", will be printed in the last column of the file name field to indicate that the name has been truncated. For ASCII and BDAT files, the number of records shown is the number of records specified in the CREATE statement. This behavior was requested by customers for compatibility with existing programs. The actual number of records may be more or less and can be determined by examining STATUS register three of an I/O Path ASSIGNed to the file. Or the file can be CREATEd with zero records; **CAT** then reports the actual number of records.

Short 8.3 File Format

The listing format for the DOS (FAT) file system is designed to be compatible with HP BASIC/DOS (Viper). The format chosen by HP is very similar to the format used for the UNIX file system. This is an example of output in DOS format:

```
DIRECTORY: C:\HTB
LABEL: DEMO
FORMAT: DOS
AVAILABLE BYTES:  34004992
              FILE   NUMBER   REC MODIFIED
FILE NAME     TYPE  RECORDS   LEN DATE        TIME PERMISSION
============ ===== ======== ===== ========= ===== ==========
HTB.KEY      BDAT         2   256 10-Oct-98 14:00  RW-RW-RW-
HP-PCL.D86   BIN       1384     1 21-Nov-95  0:00  RW-RW-RW-
HTB.PIF      DOS        545     1 24-Jul-00 11:12  RW-RW-RW-
```

The following information is given in the header. The number specifies the line number on which the information is given:

1. Path specifier (volume specifier and full path name).
2. Volume label of the device.
3. The file system type, i.e. DOS or FAT.
4. Amount of free space on the device in bytes (NOT blocks).
5. Column headings for file information.
6. Column headings for file information.

Note that HP BASIC gives the free space in *blocks*, while HTBasic gives it in *bytes*. The file information occurs in the following columns:

Column	**Information**
1-12	filename or directory name
14-18	file type, BDAT, DIR, PROG, etc.
20-27	number of records in the file
29-33	record length of each record
35-43	modification date in the form DD-MMM-YY
45-49	modification time in the form HH:MM
	File Access Permissions
52	read access - An R is always present
53	write access - A W allows write
54	execute flag - An X means executable
55-60	File Access Permissions repeated

The file type is determined in the following manner: The file type is listed as DIR for a directory and SYSTM if the file has the DOS System Attribute. If the file has an HTBasic file type header, then the file type (BDAT, ASCII, PROG or BIN) found in the header is given. If the header can't be read, then "LOCKD" is given. All other files are ordinary files and are listed with no file type or a file type of "DOS". If a file has the DOS Hidden Attribute, then the file is not listed. The DOS Archive Attribute is ignored.

See the note earlier explaining how the number of records is listed. DOS updates directory entries only when a file is closed. Thus, the length of a file will not appear to change in a **CAT** as the file is written.

The file permissions are listed as read, write and execute. To mimic UNIX, they are repeated three times. The file permissions are determined in the following manner. The read access, "R", is always set since DOS does not have a deny-read permission. The write access, "W" is set unless the DOS Read-Only Attribute is set. The execute flag is set if the file extension is ".BAT", ".COM" or ".EXE" meaning the file can be executed from the DOS command prompt.

Long Filename Format

Under later versions of DOS, and Windows, some file systems allow long names with embedded spaces. However, by default CAT still uses the FAT listing format, providing 8.3 compatible filenames. To enable display and use of long filenames, use the statement

```
CONFIGURE LONGFILENAMES ON
```

With **LONGFILENAMES ON**, spaces are not deleted from directory and file specifiers since they may be significant. It is roughly modeled after the NT DIR command. The listing format with **LONGFILENAMES ON** is given below.

```
DIRECTORY: C:\PROGRAM FILES\HTBWIN
LABEL: MYDISK
FORMAT: NTFS
AVAILABLE BYTES:  54132736
FILE    NUMBER   REC MODIFIED
TYPE   RECORDS   LEN DATE      TIME  ATTRIB FILE NAME
===== ======== ===== ========= ===== ====== ========================
DIR          0     1 26-Apr-93 14:04  D     Look at this file name
BDAT         2   256 10-Oct-89 14:00 A      HTB.KEY
BIN       1888     1 30-Dec-95 13:37 A      HP-PCL.DW6
        303967     1 25-Aug-00 10:06 A    R Data
PROG       706     1  2-Jun-05 14:52 A      AUTOST
```

Note that filenames are listed at the end. Standard DOS or NT file attributes are also presented. The information in the header is the same as for the FAT file system. The file information is presented in the following columns:

Column	**Information**
1-5	file type, BDAT, DIR, PROG, etc.
7-14	number of records in the file
16-20	record length of each record
22-30	modification date in the form DD-MMM-YY
32-36	modification time in the form HH:MM
	File Attributes:
38	"A" if Archive Attribute set
39	"D" if Directory Attribute set
40	Always " "
41	"S" if System Attribute set
42	"H" if Hidden Attribute set
43	"R" if Read-only Attribute set
45-	filename or directory name

Column 40 will always be blank, since files with the Hidden Attribute are not listed.

Long Year Format

The listing format for **CAT** may be changed to display a four-digit year. Using the command CONFIGURE LONGCATDATES ON will display four digits for the year instead of two. To display only two digits, use CONFIGURE LONGCATDATES OFF. These settings may also be set in the Run Environment Dialog box. It is impoirtant to rember that when using the four-digit year to dimension array length to accomidate the extra two characters. Typically this means dimensioning the array to hold 62 rather than 60 characters when sending the CAT to an array.

SRM Format

When **CAT** is directed to a string array, it produces the SRM catalog format. The elements of the array must be declared to contain at least 80 characters. If the array has more elements than necessary, the extra elements are set to zero length. If the array doesn't have enough elements, information about the additional files is thrown away and no error is reported.

The SRM listing format is compatible with HP BASIC, for compatibility with existing programs. This is an example of output in SRM format:

```
DIRECTORY: C:\HTB
LABEL: DEMO
FORMAT: DOS
AVAILABLE BYTES:  33939456
                      SYS  FILE   NUMBER   RECORD     MODIFIED     PUB OPEN
FILE NAME        LEV TYPE  TYPE  RECORDS   LENGTH DATE         TIME ACC STAT
================ === ==== ===== ======== ======== ================ === ====
HTB.KEY            1 HTB   BDAT         2      256 10-Oct-89  14:00 MRW
HP_DATA            1 S300  BDAT       384      256 21-Nov-91   0:00 MRW
HTB.PIF            1       DOS        545        1 24-Jul-99  11:12 MRW
```

The following information is given in the header. The number specifies the line number on which the information is given:

1. Path specifier.
2. Volume label of the device.
3. Name of the file system, not the catalog format, i.e. DOS, UNIX, etc.
4. Amount of free space on the device in bytes (NOT blocks).
5. Column headings for file information.
6. Column headings for file information.

Note that the path specifier is preceded by the word "DIRECTORY:". This is different from HP BASIC. Also, line three gives the name of the file system, not the name of the listing format. In line four, HP BASIC gives the free space in *blocks*, while HTBasic gives it in *bytes*. The file information occurs in the following columns:

Column	**Information**
1-21	filename or directory name
23-25	level: always 1
27-30	system type: HTB, S300, S500 or blank
32-36	file type, BDAT, DIR, PROG, etc.
38-45	number of records in the file
47-54	record length of each record
56-64	modification date in the form DD-MMM-YY
67-71	modification time in the form HH:MM
	File Access Permissions
73	Manager access - If an M is present then anyone can read, write, PURGE
74	Read access - An R allows read
75	Write access - A W allows write
77-80	Open status, OPEN, LOCK, CORR or blank

See the notes earlier about file names and number of records. The Level is always listed as 1. The system type specifies the kind of file header for typed files. If the file header is an HTB header, the system type is "HTB". If the file header is an HP LIF header, the system type is "S300" (or "S500" in the special case of Series 500 BDAT files). If the file is an ordinary file, the system type column is blank, since the file has no header. File access permissions are mapped into SRM permissions in a logical manner from the actual operating system permissions. In general, the Open Status is undefined.

Selecting a sub-set of files to be displayed

There are three ways to select a subset of files in a directory to be displayed. The first method is to use the **SKIP** option: specify that the first N files are not sent to the destination. The second method is to use the **SELECT** option: specify in the **SELECT** string the beginning characters of the files you wish listed, all files that don't begin with the selected characters are not displayed.

The last method of selecting files is to use wildcards. The media specifier, *source*, is expanded to include a file name template including wildcards. See WILDCARDS for an explanation of how to use wildcard characters. Wildcarding is always enabled for the CAT statement. WILDCARDS OFF has no effect.

The following examples illustrate the last two methods of selecting files. Shown side-by-side are examples which select the same sub-set of files to be displayed. The example on the left uses the wildcard style of selection, while the example on the right uses the **SELECT** option.

Wildcard style	**SELECT style**
CAT "H*"	CAT ;SELECT "H"
CAT "TEXT.*"	CAT ;SELECT "TEXT."
CAT "A:R*"	CAT "A:";SELECT "R"
CAT "\DOS\BASICA.C*"	CAT "\DOS";SELECT "BASICA.C"

Do not use both these methods at once. If you wish to specify a wildcard, use the wildcard style.

The following are examples of commands which can only be done using the wildcard style.

```
CAT "*.BAS"! List only files with the .BAS extension
CAT "A?C"  ! List files with 1st letter "A", any second
           ! letter and 3rd letter "C".
```

COUNTing the number of lines displayed

If the **COUNT** option is included, the variable is assigned the number of lines that was sent to the destination. This can be especially useful when sending the output to a string array for later processing. Except for PROG files, the count includes the header, files that are **SKIP**ped, files actually sent to the destination, files not sent to a string array because the array was too small and the "AVAILABLE ENTRIES" line of a catalog of a PROG file.

Suppressing the header

If the **NO HEADER** option is included, then just the files are sent to the destination and **COUNT** accounts for no header lines. For catalogs of a PROG file, the "AVAILABLE ENTRIES" line is also suppressed.

Listing filenames only

If the **NAMES** option is included, then only filenames are listed. Both the header and other file information is suppressed. If output is directed to a device, names are output in five columns. If output is directed to a string array, output is one name per element. The CAT statement executes considerably faster with this option.

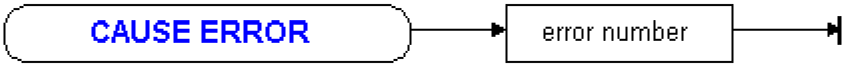
See Also:

COPY, CREATE, CREATE ASCII, CREATE BDAT, MASS STORAGE IS, PERMIT, PROTECT, PURGE, RENAME, SYSTEM$("MSI")

CAUSE ERROR

Simulates a specified error.

Syntax: CAUSE ERROR error-number



Sample:

```
CAUSE ERROR Err
IF Testing THEN CAUSE ERROR 80
```

View Sample: CAUSE ERROR.BAS **(also found in examples directory)**

Description:

When the statement is executed, it is as though the error specified actually occurred and the normal error related functions are affected: ERRL, ERRLN, ERRM$ and ERRN. **CAUSE ERROR** is useful in debugging error handlers.

See Also:

CLEAR ERROR, ERRL, ERRLN, ERRM$, ERRN, ERROR RETURN, ERROR SUBEXIT, OFF ERROR, ON ERROR

CHANGE

Finds and replaces strings.

Syntax: CHANGE old TO new [IN first-line [,last-line]] [;ALL]

where: old and new = string-literals
first-line and last-line = line-number | line-label

Sample:

```
CHANGE "Apples" TO "Oranges" IN 1200,1500
CHANGE "Delete this sentence." TO ""
CHANGE "1988" TO "1989";ALL
CHANGE "unquoted" TO """quoted"""
```

Description:

The **CHANGE** statement is an editor command that allows you to search and replace character sequences. The old and new string literals are used exactly as given with the case being significant.

The **CHANGE** command from the HTBasic Windows Editor input line brings up the **FIND** window and fills fields with old and new values. All other options are ignored.

If **ALL** is included in the **CHANGE** statement, then all changes are made automatically. If **ALL** is not specified, the computer searches for each occurrence, replaces the item, displays the line with the change and then asks you if you want this replacement. If you do, press ENTER; if you don't, press CONTINUE. If you wish to abort the **CHANGE** statement, press any other function key. When no further occurrences of the search string can be found a message "*old* not found" is displayed.

CHANGE is not allowed while a program is running, but it may be used when the program is paused. An error message will be displayed if a syntax error occurred during any **CHANGE** operation. When the line is corrected the **CHANGE** command continues. The **CHANGE** operation is aborted if a change exceeds the maximum allowable length of a program line or if a line number is altered.

If first-line doesn't exist, the line immediately after that line number is used. If a non-existent line label is specified, an error will be reported. If last-line is specified, searching will end with that line. If the line doesn't exist, the line immediately before that line number is used. If a non-existent line label is specified, an error will be reported. If last-line is not specified, searching will end with the last line in the program. This command can only be executed from the keyboard. It cannot be included in a program.

See Also:

COPYLINES, DEL, DELSUB, EDIT, FIND, INDENT, MOVELINES, REN, SECURE, XREF

CHECKREAD

Enables/disables verification of data sent to disk.

Syntax:

CHECKREAD ON
CHECKREAD OFF

Sample:

```
If Vital THEN CHECKREAD ON
CHECKREAD OFF
```

Description:

This command enables or disables verification of data sent to the mass storage media. If the data that is written fails to verify correctly, an error is reported. **CHECKREAD ON** enables and **CHECKREAD OFF** disables verification. The method of verification depends on the operating system and hardware of your computer. If the operating system does not support verification, this statement is ignored.

CHECKREAD is not supported by HTBasic and is ignored.

CHGRP

Sets the Group Ownership of a file.

Syntax: CHGRP group, file-specifier

where: group = numeric-expression rounded to an INTEGER

Sample:

```
CHGRP 32,"/usr/users/Florece/file1"
CHGRP 0,"/etc/passwd"
```

Description:

On operating systems which support both group and individual ownership of a file, **CHGRP** changes the group associated with a file. If the operating system does not support this call or if you do not have the proper privilege to change the group, an error is returned when the statement is executed. However, under any version of HTBasic, the editor will allow this statement to be entered and the syntax checker will check it for correctness.

The FAT file does not support file ownership. **CHGRP** is not supported by HTBasic, executing this statement will cause an error.

See Also:

CHOWN, CREATE, PERMIT, TIMEZONE IS

CHOWN

Sets the Individual Ownership of a file.

Syntax: CHOWN id, file-specifier

where: id = numeric-expression rounded to an INTEGER

Sample:

```
CHOWN 512,"/usr/users/Sherry/file2"
CHOWN 0,"/dev/tty1"
```

Description:

On operating systems which support individual ownership of a file, **CHOWN** changes the ownership of a file. If the operating system does not support this call or if you do not have the proper privilege to change the ownership, an error is returned when the statement is executed. However, under any version of HTBasic, the editor will allow this statement to be entered and the syntax checker will check it for correctness.

The FAT file system does not support file ownership. **CHOWN** is not supported by HTBasic, executing this statement will cause an error.

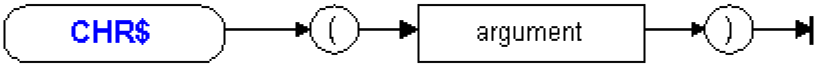
See Also:

CHGRP, CREATE, PERMIT, TIMEZONE IS

CHR$

Creates ASCII character from decimal value.

Syntax: CHR$(numeric-expression)



Sample:

```
Lf$=CHR$(10)
Lowr$=CHR$(NUM(Uppr$)+32)
A$=CHR$(65)
```

View Sample: **CHR$.BAS** **(also found in examples directory)**

Description:

The argument of the **CHR$** function is a numeric expression which is rounded to an integer. A value within the range 0 to 255 is then extracted from the integer by using the low-order byte of the 16-bit word. The ASCII character which corresponds to this value is assigned to the specified string variable. Only one character is assigned to the target string. An ASCII table is included in Appendix B.

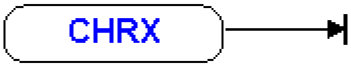
See Also:

NUM

CHRX

Returns the width of a character cell.

Syntax: CHRX



Sample:

```
X1=CHRX
ALLOCATE INTEGER Charcell(1:CHRY,1:CHRX)
PRINT “Your Characters are “:CHRX&” Wide”
```

View Sample: **CHRX.BAS** **(also found in examples directory)**

Description:

If your computer display supports multiple display modes or fonts having different character widths, the value returned by **CHRX** is the width for the current display mode.

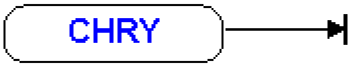
See Also:

CHRY, SET CHR

CHRY

Returns the height of a character cell.

Syntax: CHRY



Sample: CHRY

```
ALLOCATE INTEGER Charcell(1:CHRY,1:CHRX)
PRINT “Your Characters are “:CHRY&” High”
```

View Sample: **CHRY.BAS** **(also found in examples directory)**

Description:

If your computer display supports multiple display modes or fonts having different character heights, the value returned by **CHRY** is the height for the current display mode.

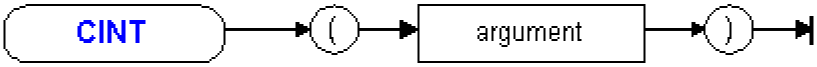
See Also:

CHRX, SET CHR

CINT

Converts a value to INTEGER.

Syntax: CINT (numeric-expression)



Sample: `OUTPUT @I;CINT(X*1.1)`

View Sample: **CINT.BAS** **(also found in examples directory)**

Description:

The **CINT** function is useful for forcing the type of a variable or value to INTEGER. For example, suppose you are writing binary integers to a file and one value must be multiplied by 1.1 before being written. X*1.1 gives a REAL result, which outputs eight bytes to the file. Even INT(X*1.1) gives a REAL. **CINT**(X*1.1) forces the value to be INTEGER and two bytes are written to the file.

Notice the differences among **CINT**, FIX and INT. **CINT** converts a REAL value to an INTEGER value by substituting the closest INTEGER to the value. FIX returns the closest integral value between the REAL value and zero. INT returns the closest integral value between the REAL value and negative infinity. Also, **CINT** actually changes the type from REAL to INTEGER while INT and FIX return integral results without changing the type. The following table helps illustrate these differences:

Value x	CINT(x)	FIX(x)	INT(x)
2.6	3	2.0	2.0
2.2	2	2.0	2.0
-2.2	-2	-2.0	-3.0
-2.6	-3	-2.0	-3.0

Porting to HP BASIC:

CINT is a new HTBasic function that is not available in HP BASIC. It should not be used in programs that must be ported back to HP BASIC.

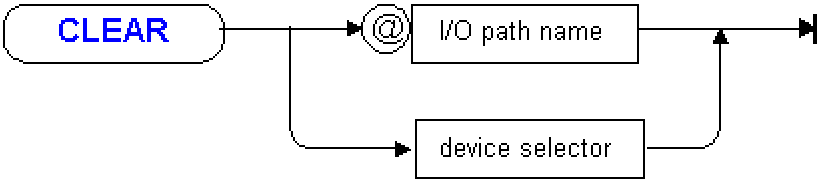
See Also:

DROUND, FIX, FRACT, INT, PROUND, REAL

CLEAR

Sends an IEEE-488 bus Device Clear.

Syntax: CLEAR { device-selector | @io-path }



Sample: `CLEAR 7`

```
CLEAR 701
CLEAR Adevice
CLEAR @Path
```

Description:

CLEAR causes the active controller to send a Device Clear to one or more devices. The effect on the device is device-dependent. If the computer is not the active controller, an error is generated. If primary addressing is specified the bus action is: ATN, MTA, UNL, LAG, SDC. If only an interface select code is specified the bus action is: ATN, DCL.

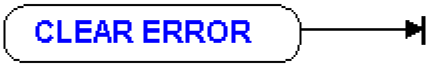
See Also:

ABORT , LOCAL, PASS CONTROL, PPOLL, REMOTE, REQUEST, SEND, SPOLL, TRIGGER

CLEAR ERROR

Resets all error indicators.

Syntax: CLEAR ERROR



Sample:

```
CLEAR ERROR
IF Finish THEN CLEAR ERROR
```

View Sample: **CLEAR ERROR.BAS** **(also found in examples directory)**

Description:

CLEAR ERROR resets ERRL, ERRLN, ERRM$ and ERRN to their default start-up values.

See Also:

CAUSE ERROR, ERRL, ERRLN, ERRM$, ERRN, ERROR RETURN, ERROR SUBEXIT, OFF ERROR, ON ERROR

CLEAR LINE

Clears the keyboard input line.

Syntax:

CLEAR LINE

Sample: `IF Signal THEN CLEAR LINE`

View Sample: **CLEAR LINE.BAS** **(also found in examples directory)**

Description:

This command is equivalent to pressing the CLR LN key and replaces the non-intuitive command: OUTPUT KBD;CHR$(255)&"#";.

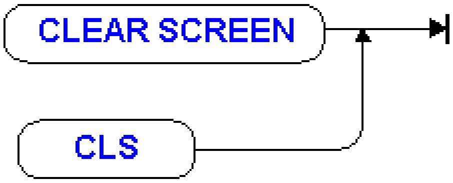
See Also:

CLEAR SCREEN

CLEAR SCREEN

Clears the ALPHA display.

Syntax: CLEAR SCREEN
CLS



Sample: `IF Ready THEN CLEAR SCREEN`

View Sample: **CLEAR SCREEN.BAS** **(also found in examples directory)**

Description:

CLS is an abbreviated form of **CLEAR SCREEN**. This command is equivalent to pressing the CLR SCR key and replaces the non-intuitive command: OUTPUTKBD;CHR$(255)&"K";.

On bit mapped displays with MERGE ALPHA WITH GRAPHICS in effect, this command will also clear the graphic screen.

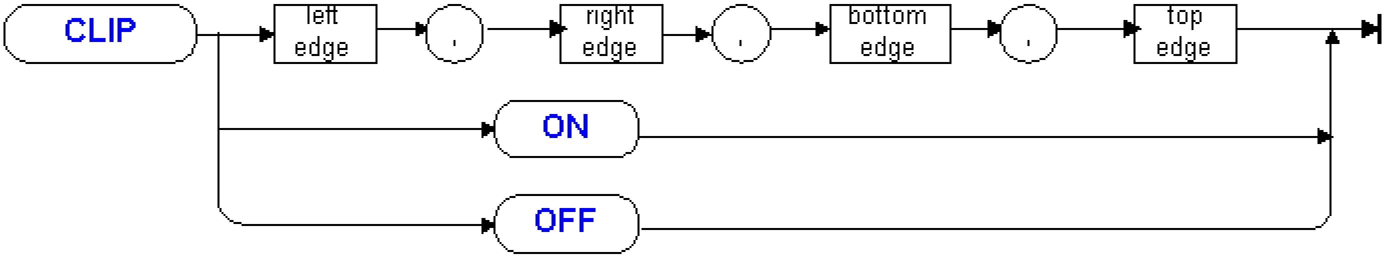
See Also:

CLEAR LINE

CLIP

Changes the clipping rectangle.

Syntax: CLIP left,right,bottom,top
CLIP ON
CLIP OFF



Sample: `CLIP 10,20,5,25`

View Sample: **CLIP.BAS (also found in examples directory)**

Description:

CLIP changes the clipping rectangle. Lines, areas and labels are clipped so that portions outside the clipping rectangle are not displayed. The PLOTTER IS statement sets the clipping rectangle to the hard-clip limits (which are the user specified values or the maximum allowed by the device or page size). The VIEWPORT statement sets the clipping rectangle to the edge of the VIEWPORT.

When values are specified with the **CLIP** statement, the clipping rectangle is set to the values specified. The units used are WINDOW (or SHOW) units, not VIEWPORT units.

The **CLIP OFF** statement sets the clipping rectangle back to the hard-clip limits. The **CLIP ON** statement restores the clipping rectangle to the last clipping rectangle set up by **CLIP** or VIEWPORT. If no **CLIP** or VIEWPORT has been executed, **CLIP ON** sets the clipping rectangle to the hard-clip limits.

Execute **CLIP** to add labels, comments, graphics or any other plotting that is to be done outside the VIEWPORT (assuming the VIEWPORT is less than the hard-clip limits).

See Also:

CLEAR SCREEN, DRAW, MOVE, PLOT, POLYGON, POLYLINE, SHOW, VIEWPORT, WINDOW

CLS

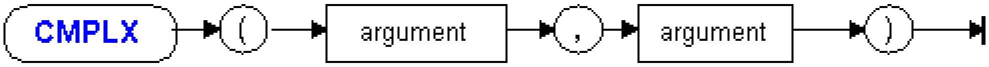
See CLEAR SCREEN.

CLEAR SCREEN

CMPLX

Combines real and imaginary parts to return a complex number.

Syntax: CMPLX(numeric-expression, numeric-expression)



Sample: `PRINT Z*CMPLX(0,1)`

```
Z=CMPLX(X,Y)
```

View Sample: **CMPLX.BAS** **(also found in examples directory)**

Description:

This function allows a complex number to be assembled from two numeric expressions. The first expression specifies the real part and the second specifies the imaginary part. This function also allows complex constants, such as **CMPLX**(PI,6.7), to be expressed in a program.

To assemble a complex number from magnitude and angle rather than real and imaginary parts, use this method:

```
Z = CMPLX( Magnitude*COS(Angle), Magnitude*SIN(Angle))
```

If a complex number is used as an argument to **CMPLX**, then only the real part of the argument is used. For example, **CMPLX**(**CMPLX**(1,2), **CMPLX**(3,4)) is equal to **CMPLX**(1,3).

See Also:

ABS, ARG, CONJG, IMAG, REAL

COLOR

Defines and selects the color for graphics.

Syntax:

AREA COLOR h, s, l
AREA INTENSITY r, g, b
AREA PEN pen-number
PEN pen-number
SET PEN pen-number COLOR h, s, l
SET PEN pen-number COLOR numeric-array(*)
SET PEN pen-number INTENSITY r, g, b
SET PEN pen-number INTENSITY numeric-array(*)

where:

h,s,l, r,g,b = each is a numeric-expression in the range zero to one.
pen-number = see below.

Sample:

```
SET PEN 1 COLOR H,S,L
AREA INTENSITY R,G,B
AREA PEN 2
SET PEN Num COLOR H,S,L
PEN 1
```

View Sample: COLOR.BAS **(also found in examples directory)**

Description:

Specifying a Color using the HSL system

Use the keyword **COLOR** to specify a color in the HSL (Hue, Saturation, Lightness) color space. The HSL color space is designed to be intuitive and follows the model of mixing paints. An artist preparing a color for a painting first selects a hue (pure color pigment). He may then add black or white paint to arrive at the desired color. Adding white serves to wash out the color. In scientific terms, we say this affects the "saturation" of the color. The artist may then adjust the brightness by adding black paint. This affects the amount of light reflected by the pigment. We call this the luminosity.

Saturation ranges from zero (white) to one (pure color - no added white). Luminosity ranges from zero (black) to one (pure color - no added black). Hue ranges from zero to one. The following table gives an indication of where several colors occur in that range:

Hue	Value
Red	.000
Yellow	.167
Green	.333
Cyan	.500
Blue	.667
Magenta	.833
Red	1.00

Specifying a Color using the RGB system

Use the keyword **INTENSITY** to specify a color using the RGB (Red, Green, Blue) color space. The RGB color space is designed to match the way in which our eyes work and in turn, the way in which television and computer displays are designed. The display has three color guns: Red, Green and Blue. By specifying a number in the range zero (corresponding to zero intensity) to one (corresponding to maximum intensity) for each of the three guns, you can uniquely define all the colors which can be produced by that display.

Pen Numbers

A computer display system is limited in the number of different colors it can display at the same time. If N is the number of different colors which can be displayed simultaneously, then legal pen numbers are the integers 0 to N-1.

Drawing Mode Table

The writing mode of the pen is specified by the current drawing mode and the sign of the pen number. GESCAPECRT,4 is used to change to normal drawing mode. GESCAPECRT,5 is used to change to alternate drawing mode. The following table defines the different writing modes available. P is a positive pen number, X is the present value of a pixel.

	GESCAPE CRT,4	**GESCAPE CRT,5**
Statement	**Normal**	**Alternate**
PEN P	P	BINIOR(X,P)
AREA PEN P	P	BINIOR(X,P)
PEN 0	BINCMP(X)*	0
AREA PEN 0	0	0
PEN -P	BINAND(X,BINCMP(P))	BINEOR(X,P)
AREA PEN -P	BINAND(X,BINCMP(P))	BINAND(X,BINCMP(P))

*PEN 0 in Normal Drawing Mode will do BINCMP(X) in non-color map mode and 0 in COLOR MAP mode.

Pen Numbers

The SET PEN statement explains pen color assignments. The following table gives the default color to pen assignments.

PEN	**COLOR**
0	black
1	white
2	red
3	yellow
4	green
5	cyan
6	blue
7	magenta
8	black
9	olive green
10	aqua
11	royal blue
12	maroon
13	brick red
14	orange
15	brown

See Also:

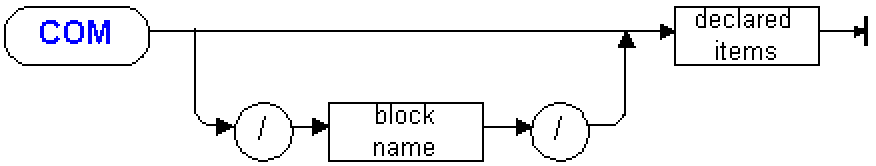
AREA, GESCAPE, PLOTTER IS, PEN, SET PEN

COM

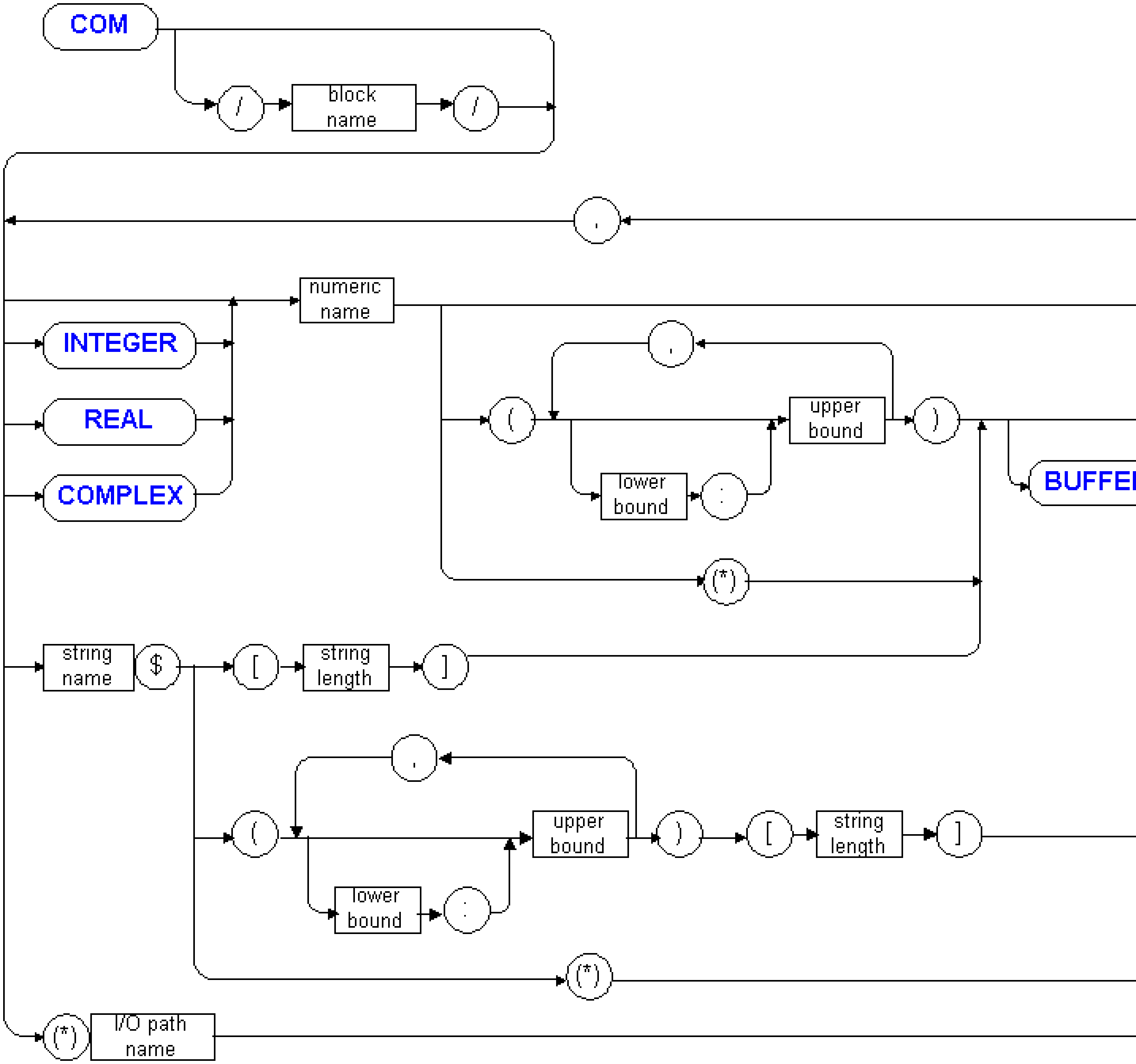
Defines global variables.

Syntax: COM [/ com-block-name /] item [,item...]

where: item = [type] numeric-name [{(bounds)|(*)} [BUFFER]] |
string-name$ [[length]] [BUFFER] |
string-name$ { (bounds) [[length]] | (*) } |
@io-path
type = {REAL | INTEGER | COMPLEX}
bounds = [lower-bound :] upper-bound [,bounds...]
upper bound, lower bound and length = integer constants



expanded diagram:



Sample: `COM P1,Fft$[1024] BUFFER`

```
COM INTEGER I(5),REAL Array(-365:364)
COM /Block/ Name$,@Source,INTEGER Cross(*)
```

View Sample: COM.BAS (also found in examples directory)

Description:

COM allocates a block of memory where variables can be held in "common" between one or more program contexts. Any subprogram or main context can access a "common" variable by including a **COM** statement which references the correct block of memory. One unnamed **COM** block is provided. To reference it, leave off the block name. The unnamed **COM** block must be declared in the main context. All other **COM** blocks are referenced by name. The name is global to all contexts.

Declaring a COM block

A **COM** block may contain so many variables that it takes several lines to declare them all. As long as all the **COM** statements are in the same context and all reference the same block name (or all have no block name), it is completely legal to divide the **COM** block declaration onto several lines. The following is an example:

```
COM /Block1/ Var1,Var2
COM /Block1/ Var3,Var4
```

Furthermore, the statements don't have to be next to each other. In fact, statements declaring two or more **COM** blocks can be intermixed. The **COM** statements must preceed any OPTION BASE statement that is present.

Parameters are not allowed in **COM** statements. Numeric variables are considered REAL until an **INTEGER** declaration is seen. Variables are then considered **INTEGER** until a **REAL**, I/O path or string is declared. String variables must have their length declared when declared in a **COM** block. Buffer variables are declared by specifying **BUFFER** after each variable's name. **BUFFER** variables are used with the TRANSFER statement.

The maximum number of array dimensions is six and the lower bound must be less than or equal to the upper bound value. In the first context that an array or string is declared, the **COM** statement must explicitly specify array subscript bounds and string lengths. In subsequent contexts, **COM** statements need only specify the string name or the array name with a full array specifier "(*)".

Matching COM blocks

The **COM** blocks in each context must match. In a given **COM** block, the individual variable names do not have to match, but the number of variables and their type must agree. The boundaries of arrays do not have to be the same, but the RANK (number of dimensions) and the SIZE must match.

Creation and Deletion of COM blocks

COM variables have a different lifetime than normal variables. When a **COM** block is created, the variables are all initialized to zero (or zero length strings). The variables then exist and retain values assigned to them until the **COM** block is deleted.

A **COM** block is initially created when a program context is "prerun" and the context declares a **COM** block that does not already exist. A prerun will be done when you:

Press RUN or STEP when no program is running
Execute the RUN command when no program is running
Execute GET or LOAD from a program
Execute GET or LOAD command that begins program execution

During prerun, if a **COM** block is declared which already exists, the new and old declarations are compared for compatibility. If they are found to be compatible, then the **COM** block is left untouched and the variables retain their previous values. If they are found to be incompatible then an error is returned. If a REDIM can make arrays compatible, then the arrays will be REDIMed. A **COM** block exists until a SCRATCH A or SCRATCH C deletes it. Even if you delete the program which refers to a **COM** block, it remains in memory until a SCRATCH A or C is executed.

When you LOAD a new program, all **COM** blocks in memory will be checked against the **COM** blocks defined in the new program and any unreferenced **COM** blocks will be deleted.

See Also:

ALLOCATE, DIM, INTEGER, OPTION BASE, REAL, REDIM, TRANSFER

COMMAND$

Returns a copy of the command line.

Syntax: COMMAND$

Sample:

```
PRINT "Switches: "&COMMAND$
C$[4;10]=LWC$(COMMAND$)
```

View Sample: **COMMAND$.BAS** **(also found in examples directory)**

Description:

The **COMMAND$** function returns the command line used to start HTBasic, including any command line options specified.

HTBasic returns the entire command line. This is useful if symbolic links are made to the HTBasic executable and the AUTOST program wishes to react differently depending on the name used to start HTBasic.

Porting to HP BASIC:

COMMAND$ is a new HTBasic function that is not available in HP BASIC. It should not be used in programs that must be ported back to HP BASIC.

See Also:

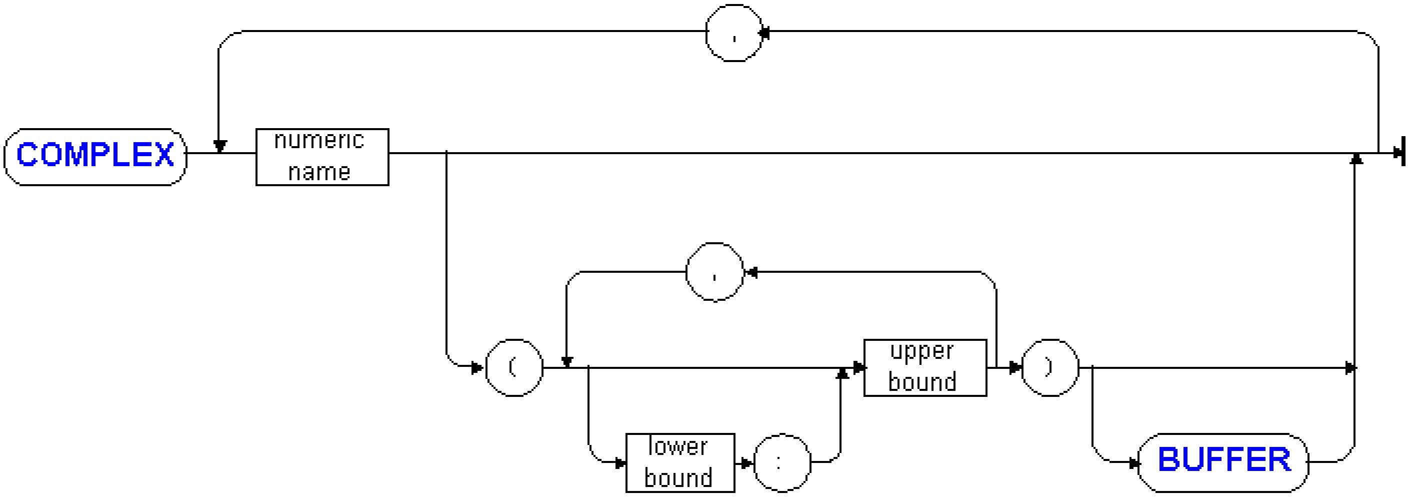
ENVIRON$, EXECUTE, SYSTEM$

COMPLEX

Reserves storage for complex variables and arrays.

Syntax: COMPLEX variable [,variable...]

where: variable = numeric-name [(bounds) [BUFFER]]
bounds = [lower-bound :] upper-bound [,bounds]
lower/upper-bound = integer constant in the range -32768 to 32767.



Sample: `COMPLEX Z, C(-10:10,4)`
`COMPLEX Tx(512) BUFFER`

View Sample: **COMPLEX.BAS** **(also found in examples directory)**

Description:

COMPLEX declares, dimensions and reserves memory for complex variables and arrays. **COMPLEX** variables use sixteen bytes of storage space. An array's maximum dimension is six and each dimension can hold a maximum of 32,767 elements. If a lower bound is not specified, the default is the OPTION BASE value (0 or 1). A **COMPLEX** variable may be declared a buffer by specifying **BUFFER** after the variable name. Buffer variables are used with the TRANSFER statement.

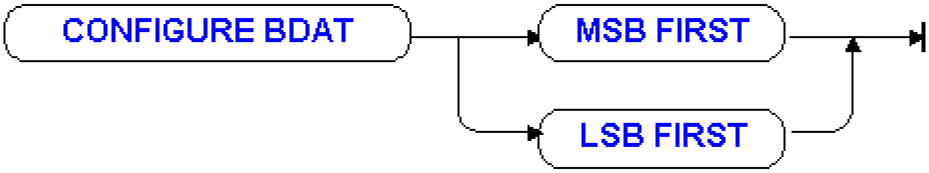
See Also:

ALLOCATE, COM, DEF FN, DIM, INTEGER, REAL, SUB, TRANSFER

CONFIGURE BDAT

Specifies the byte order for CREATE BDAT.

Syntax: CONFIGURE BDAT {MSB | LSB} FIRST



Sample:

```
CONFIGURE BDAT MSB FIRST
CONFIGURE BDAT LSB FIRST
```

View Sample: **CONFIGURE BDAT.BAS** **(also found in examples directory)**

Description:

CONFIGURE BDAT specifies the byte ordering to use with each BDAT file created after this statement is executed. By default, BDAT files are created with the same byte order as the computer. The IBM PC and compatibles use **LSB FIRST**. The Sun SPARCstation and HP Series 700 use **MSB FIRST**. Since HP BASIC can only use **MSB FIRST** files, if you wish to CREATE BDAT files on a PC which can be used by an HP BASIC workstation, you must use **CONFIGURE BDAT MSB FIRST** before creating the files. HPCOPY will print a warning when it copies any BDAT file with **LSB FIRST** byte ordering.

BDAT files created with HP file headers are always created **MSB FIRST**, regardless of the setting of this statement. See CONFIGURE CREATE.

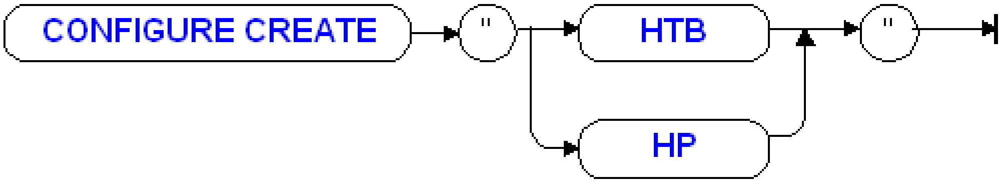
See Also:

CONFIGURE CREATE, CONFIGURE SAVE, CREATE BDAT

CONFIGURE CREATE

Specifies the kind of file header used with typed files.

Syntax: CONFIGURE CREATE {"HP" | "HTB"}



Sample: `CONFIGURE CREATE "HP"`
`CONFIGURE CREATE "HTB"`

View Sample: **CONFIGURE CREATE.BAS** **(also found in examples directory)**

Description:

CONFIGURE CREATE specifies the kind of file header to use when creating a LIF ASCII or BDAT file. By default, HTBasic creates "HTB" file headers, since they are two or three times smaller than HP LIF headers. BDAT files with HTB headers can also be created with data in either LSB or MSB byte ordering (see CONFIGURE BDAT). File operations are much faster when the byte ordering of the file matches the byte ordering of the computer. Files with HTB file headers, when copied with HPCOPY, are completely compatible with HP BASIC.

Use **CONFIGURE CREATE** "HP" if you wish to create data files that are simultaneously accessed over a network by HTBasic and HP BASIC. Files with HP LIF headers can also be "binary" copied among DOS or UNIX media for access by the HP Language Coprocessor (Viper card), HP BASIC and HP BASIC/UX.

HTBasic can always use files with either header, regardless of the setting of **CONFIGURE CREATE**. The setting affects file creation only. A CAT listing in SRM format shows the kind of file header of each file in the System Type column.

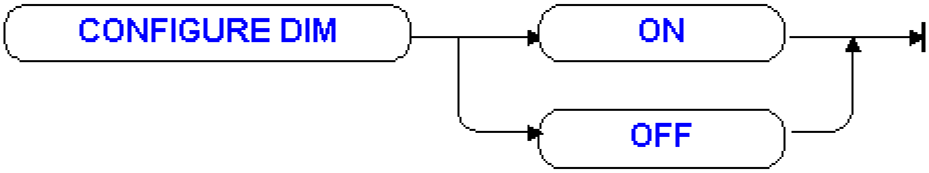
See Also:

CONFIGURE BDAT, CONFIGURE SAVE, CREATE BDAT

CONFIGURE DIM

Turns implicit variable dimensioning on or off.

Syntax: CONFIGURE DIM { ON | OFF }



Sample: `CONFIGURE DIM ON`
`CONFIGURE DIM OFF`

View Sample: **CONFIGURE DIM.BAS** **(also found in examples directory)**

Description:

CONFIGURE DIM turns implicit variable and string dimensioning on or off. By default it is on and if a variable is never declared, it is assumed to be REAL. If a string is never declared, it is assumed to have a maximum length of 18. If an array is never declared, it is implicitly declared having the number of subscripts found in its first occurrence, with each dimension having the default OPTION BASE lower bound and an upper bound of ten.

When **CONFIGURE DIM** is **OFF**, then each variable, string and array must be explicitly declared using REAL, INTEGER, LONG, COMPLEX, and DIM statements.

During prerun, any undeclared variables generate an error message that is written to the message line. To see all these error messages turn PRINTALL IS on during prerun. If a program has already been prerun, CONFIGURE DIM OFF will not report any undeclared variables until another prerun occurs. To force a prerun to occur, change a program line and press the STEP key.

While most structured programming languages force explicit variable declaration, traditional BASIC has always allowed implicit declarations. For example, in the program:

```
20  Xyz=1
30  PRINT Xy
40  END
```

the variables Xyz and Xy are used without declaration. Many advocates of structured programming, however, feel that explicit variable declaration is preferable. Suppose that "Xy" in line 30 is a typographical error that should have been "Xyz." This type of program error is extremely difficult to find and correct in a large program. With **CONFIGURE DIM OFF**, the above program would require an additional line:

```
10  REAL Xyz
```

and the "Xy" in line 30 would be flagged as an error when you attempted to run the program.

See Also:

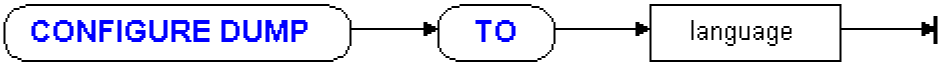
COMPLEX, DIM, INTEGER, OPTION BASE, REAL, LONG

CONFIGURE DUMP

Specifies the graphic printer language for DUMP.

Syntax: CONFIGURE DUMP TO language

where: language = string expression naming the printer language and driver options



Sample:
```
CONFIGURE DUMP TO "PCL"
```

View Sample: **CONFIGURE DUMP.BAS** **(also found in examples directory)**

Description:

CONFIGURE DUMP specifies what graphic printer language the DUMP statement uses. The language string expression specifies the name of a driver. When **CONFIGURE DUMP** is specified, dumps are directed to that driver. It is recommended that **CONFIGURE DUMP** statements be included in your AUTOST file to load any necessary drivers.

The following information is for reference only. See the *Installing and Using* manual for more specific information. The following table lists the drivers available at the time of this manual printing.

Name	**For these printers**
PCL	Advanced HP-PCL driver
PS-DUMP	Postscript printers, devices and files
GIF	Graphic Interchange Format files
WIN-DUMP	Send the dump to the default Windows printer

As an example, if you wish to use an HP LaserJet III for screen dumps on ISC 26, use the following command to change to the HP printer control language:

```
DUMP DEVICE IS 26
CONFIGURE DUMP TO "PCL"
```

If a DUMP is made before doing a **CONFIGURE DUMP**, HTBasic automatically loads and uses the WIN-DUMP driver.

Number of Colors

The number of colors in the DUMP depends on both the dump driver and the display driver. All *dump* drivers support black and white dumps. Some dump drivers can also handle 16 or 256 colors. The same is true of *display* drivers. If both the display and dump drivers support 256 colors, the dump is made in 256 colors. Otherwise if both support 16 colors, the dump is made in 16 colors. Otherwise, the dump is made in black and white.

Options

It is sometimes necessary to specify options to the drivers. Options are included by appending a semicolon to the driver name, followed by the options. The following specific driver sections contain more details on these options.

PCL Driver

The PCL dump driver provides support for devices and software that accept the Hewlett-Packard PCL printer language. The driver supports both DUMP ALPHA and DUMP GRAPHICS from bitmapped displays.

The PCL driver is loaded with a line like

```
CONFIGURE DUMP TO "PCL[;options]"
```

Options

The options are listed after the semicolon in the driver name, within the quotes. If more than one option is specified, the option names are separated by commas. When no options are given, output from the PCL driver is the same as the HP-PCL driver. The options are as follows:

ADJUST. Certain display adapters common in the PC environment use pixels that have different sizes in the horizontal and vertical directions. All pixels are considered to be square and the dump is made using the aspect ratio of the window running HTBasic.

BW. This option tells the printer to dump using white for the areas on the screen that were drawn using PEN 0 and black for the areas drawn with any other PEN. This option is the default; it need not be specified explicitly.

COLOR, CCMY, C16, and C256. These options cause the dump to be done in color to a color printer. The COLOR option uses the printer's default 8-color solid-color palette (black, white, red, green, blue, cyan, magenta, and yellow), mapping each color on the screen to the closest one from the palette. COLOR uses the default RGB palette to dump the screen; CCMY uses the default CMY palette. The C16 and C256 options use a 16- or 256-color palette on the printer, and only work with printers that have settable color palettes such as the PaintJet series and the DeskJet 1200C. With printers that use dithering to print mixed colors, you may have to specify a coarser resolution than the printer is capable of in order to enable the dithering; for example, on the original PaintJet printer, C16 and DPI90 together are needed to produce dithering; C16 and DPI180 cause the printer to use only the 8 default colors when printing.

Printing using the COLOR and CCMY options swaps black and white colors when printing, unless the INVERT option is also used.

When using the solid-color palette with older PaintJet printers, the COLOR option should be used, as these printers do not support the CMY color model. The DeskJet 500C and 550C models can only generate color screen dumps with the CCMY option.

COMPRESS. The COMPRESS option specifies that the printer being used can do "packbits"-style data compression. If this option is specified, the screen dump is transmitted to the printer using fewer data bytes. The COMPRESS option can be used with all the LaserJet IIP and IIP+ printers, all LaserJet III and IV series printers, all DeskJet series printers, the PaintJet XL300 printer (but not the older PaintJets), and the DesignJet printers, as well as other brands of printers that emulate these. Note, however, that the printers with slower CPU's will print 2-4 times slower when printing compressed data, so COMPRESS may not be a good option to use with these printers.

DPI*nnn*. This option tells the driver to use *nnn* dots per inch when dumping graphics. Without this option, the printer's default resolution is used. This option is required for the GRAY option, explained below, and for the ADJUST option (available only with PC versions of the driver). The resolution specified must be one acceptable by the printer's Raster Graphics Resolution command. For most newer devices, DPI75, DPI100, DPI150, and DPI300 are the legal values for this option. Some older printers, like the Hewlett-Packard ThinkJet, don't support this option.

With the COLOR and BW options, this option controls the size of the dump, by mapping each pixel on the screen to one of the specified-sized dots on the printer; with the GRAY option, this options controls the size of the sub-pixels used to create the printed image, as explained in the GRAY option section. On the PC, this option also sets the size of the sub-pixels used to print the image when the ADJUST option is used, as explained in the ADJUST option section.

GRAY. The GRAY option causes the driver to consult the screen's color map and calculate a gray shade for each color using the NTSC grayscale equation. Screen dumps are produced using the resulting shades of gray. If the INVERT option is not also specified, white and black are reversed after the gray shade is calculated, so that lighter colors on the screen become darker colors on the printer.

When dumps are made using this option, the driver calculates the number of printer pixels, as specified in the DPI*nnn* option, required to print a single screen pixel to make a 9 x 6 3/4 inch (23 x 17 mm) plot, up to 4 x 4 printer pixels per screen pixel. The driver sets the appropriate number of printer pixels to black to represent the gray shade of the corresponding screen pixel.

The NTSC grayscale equation is

brightness = 11% blue + 59% green + 30% red

The GRAY option is ignored unless the DPI*nnn* option is also specified.

INVERT. By default, the driver makes images with black and white exchanged from the values used on the screen. If the GRAY option is used, the driver by default reverses the gray level of all pixels dumped from that seen on the display. This is often suitable for output to a printer, where printing is done with colored inks on white paper, but may not be suitable for film output devices, where an exact image of the screen is wanted. The INVERT option causes the colors or gray levels to be dumped exactly as they are on the screen.

RELATIVE. Normally, the driver begins each dump at the left margin. The RELATIVE option causes the driver to begin each dump at the printers current print position.

EJECT. Normally, the driver ejects the page after a dump is finished. The EJECT option is no longer supported. Use CONTROL ISC,113;0 to disable the auto-eject, and CONTROL ISC,113;1 to re-enable it.

APPEND

If the APPEND keyword is used with the DUMP DEVICE IS command and if the dump device is a file, the driver appends dumps to the file, separated by form feeds.

ALPHA Dumps

The DUMP ALPHA command from a PC text screen produces a dump at the top of a US "A" or European A4 sized sheet of paper. The attributes of text on the screen, such as the reversed colors on the key labels, are lost in this mode.

Note that DUMP ALPHA from bitmapped screens on the PC dumps the text on the screen as graphics, and attributes are preserved in the dump.

If the APPEND keyword is used, subsequent DUMP ALPHA commands produce similar dumps, each on a separate sheet of paper.

PS-DUMP Driver

The PostScript dump driver provides support for devices and software that accept the PostScript graphics language. It provides support for both the DUMP ALPHA and DUMP GRAPHICS commands. The PostScript dump driver produces a screen image intended to be rendered on a US "A" size or European A4 size page. It scales the image so that its longest dimension fits in the shortest dimension of the paper with an adequate margin. When the EXPANDED keyword is used on the DUMP DEVICE IS statement, screen dumps change from their normal portrait orientation to landscape orientation.

The PostScript dump driver is loaded with the following statement:

```
CONFIGURE DUMP TO "PS-DUMP[;options]"
```

Options

The options are listed after the semicolon in the driver name, within the quotes. If more than one option is specified, the option names are separated by commas. The GREY and COLOR options are ignored in ALPHA dumps. The options are as follows:

BW. This option causes the driver to dump using the paper color for the areas on the screen that were drawn using pen 0 and the ink color (usually black) for the areas on the screen drawn with any other pen. This is reversed if the INVERT option is also used. The BW option need not be specified explicitly; it is the default.

GRAY. This option causes the driver to render colors on the computer screen as shades of gray on the printer. Each shade of gray is calculated using the NTSC grayscale equation:

brightness = 11% blue + 59% green + 30% red

Unless the INVERT option is used, the resulting brightness is inverted before printing, so that dark colors on the computer screen print as light colors and vice-versa.

COLOR. The COLOR option causes the driver to output a color image of the screen. The resulting PostScript screen image can only be rendered on a device that supports Level 2 PostScript or the color extensions of Level 1.

INVERT. By default, the driver makes images with black and white exchanged from the values used on the screen. If the GRAY option is used, the driver by default reverses the gray level of all pixels dumped from that seen on the display. This is often suitable for output to a printer, where printing is done with colored inks on white paper, but may not be suitable for film output devices, where an exact image of the screen is wanted. The INVERT option causes the colors or gray levels to be dumped exactly as they are on the screen.

ADJUST. Certain display adapters common in the PC environment use pixels that have different sizes in the horizontal and vertical directions. Without the ADJUST option, the driver dumps from these adapters using square pixels. This may result in an image that is too wide for its height. The ADJUST option forces the image to have a 4:3 aspect ratio regardless of its pixel size.

The APPEND Keyword

If the APPEND keyword is used in the DUMP DEVICE IS statement, the dump driver appends all dump images after the first one to the existing file as new pages. The driver inserts "%%Page" comments, used by some print spooling software, into the file at the beginning of each page. If the dumps are done in separate HTBasic sessions, the driver doesn't know which page it is on, so it starts over with page 1. This may be a problem with some spooling software. Also note that only one page can be present in a file that will be imported into a word processor document.

GIF Driver

The GIF dump driver provides support for software that accepts CompuServe Graphics Interchange Format (GIF) files. The DUMP ALPHA command dumps the alpha planes in graphics mode and the DUMP GRAPHICS command dumps the graphics planes.

When the EXPANDED keyword is used on the DUMP DEVICE IS statement, graphics screen dumps are rotated 90 degrees clockwise from their normal orientation.

The GIF dump driver is loaded with the following statement:

```
CONFIGURE DUMP TO "GIF[;options]"
```

Options

The options are listed after the semicolon in the driver name, within the quotes. If more than one option is specified, the option names are separated by commas. The BW option is ignored in ALPHA dumps. The options are as follows:

BW. The driver normally produces a 16- or 256-color screen dump when used with a color screen. The BW option causes the driver to produce a black-and-white screen dump with color screens. In this dump, pixels of color zero are dumped as black and pixels of any other color are dumped as white. (This is reversed if the INVERT option is also specified.)

INVERT. The driver normally dumps an image in the colors shown on the screen. The INVERT option causes the driver to reverse black and white in the dump. All other colors are unchanged.

The APPEND Keyword

If the APPEND keyword is used in the DUMP DEVICE IS statement, the GIF dump driver appends all dump images after the first one to the existing file. Note, however, that the screen type and colormap are stored when the first image is dumped. If the screen type or colormap changes, the dump images after the first one will not be correct. Also note that most software that uses the GIF format cannot process multiple images in one file.

WIN-DUMP Driver

The WIN-DUMP dump driver provides support for any printer supported by Windows that accepts bitmaps. The command to load the WIN-DUMP dump driver is:

```
CONFIGURE DUMP TO "WIN-DUMP[;options]"
```

If a DUMP is made before doing a CONFIGURE DUMP, HTBasic automatically loads and uses the WIN-DUMP driver.

Print Manager

The default interface select code (ISC) for DUMP DEVICE IS is 10, the WIN-PRINT interface. The WIN-DUMP driver can send dumps to any ISC that is assigned to a WIN-PRINT printer via Device Setup. If you change the DUMP DEVICE to any other interface, error 150 occurs when a DUMP is attempted. To send screen dumps to another interface, such as an IEEE-488 printer, use a different dump driver.

Because Windows is a multitasking environment in which several programs may try to print at once, Print Manager collects printer output into "jobs." Only when a job is done is it printed. Normally, the WIN-DUMP driver prints a single dump per print job. To mix text and screen dumps or multiple screen dumps on a single page, output some text to the page before doing the dump. For example,

```
10    CONTROL 10,113;0  ! set DUMP auto eject to off
20    ASSIGN @I TO 10
30    OUTPUT @I;"This is a  screen dump:"
40    OUTPUT @I
50    DUMP GRAPHICS
60    ASSIGN @I TO *
70    END
```

The various settings, such as margins and line height, made in the WIN-PRINT driver are honored by the WIN-DUMP driver. See the WIN-PRINT driver documentation in the *Installing and Using Manual* for more information.

The EXPANDED keyword in the DUMP statement is ignored. The DUMP is made in landscape or portrait mode depending on the printer settings, as explained in the *Installing and Using Manual.*

DUMP Size

By default, the screen image is scaled until it fills 100% of the width between the left and right margins. The size can be changed using GESCAPE code 39. This example sets the scaling to 20% of the width between the margins:

```
10    INTEGER S(1:1)
20    S(1)=20
30    GESCAPE CRT,39,S(*)
40    END
```

INVERT Option

By default, the driver inverts all colors in the image. Black and white are exchanged as well as other colors. This is often suitable for output to a black and white printer, where printing is done with black ink on white paper, but may not be suitable for color output devices, where an exact image of the screen is wanted. The INVERT option causes the colors or gray levels to be dumped exactly as they are on the screen.

Graphics Buffering

The DUMP statement is affected by graphics buffering. When graphics buffering is off, parts of a window that are obscured or off the edge of the screen are not dumped correctly. If the window is minimized, a dump of the icon is returned. When graphics buffering is on, the window is correctly dumped in all cases.

See Also:

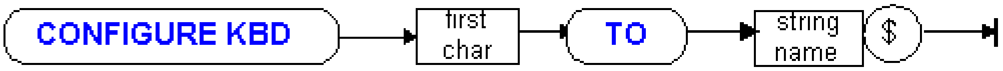
DUMP, DUMP DEVICE IS

CONFIGURE KBD

Defines keyboard mappings for character sets.

Syntax: CONFIGURE KBD first-char TO string-name$

where: first-char = numeric-expression rounded to an integer.



Sample: `CONFIGURE KBD 129 TO Mapping$`

View Sample: **CONFIGURE KBD.BAS** **(also found in examples directory)**

Description:

CONFIGURE KBD defines keyboard mappings for character sets not supported by your operating system. When in effect, **CONFIGURE KBD** substitutes characters from the given string in place of characters that come from the keyboard. This remapping is good for ASCII characters, but does not apply to function keys. (Use CONFIGURE KEY to redefine function keys.) **CONFIGURE KBD** is not intended to be a complete keyboard driver, it merely substitutes one ASCII value for another. The range of ASCII values which are remapped starts at *first-char* and extends to (*first-char* - LEN(string-name$) - 1). The string specifies the ASCII values which should be substituted for values in that range.

For example, if the keyboard is producing characters from the PC Code Page 850 character set, but the display has been set up to display the HP Roman-8 character set, the following program will cause characters from the keyboard to be translated to the display character set so that characters are displayed with the same glyphs as printed on the keyboard. If the keyboard is used to produce a character not in the HP Roman-8 character set, it is translated to CHR$(252), a solid block.

```
10    !setkbd2.bas
20    DATA  0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
30    DATA  16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
40    DATA  32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47
50    DATA  48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63
60    DATA  64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79
70    DATA  80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95
80    DATA  96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111
90    DATA  112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127
100   DATA  180,207,197,192,204,200,212,181,193,205,201,221,209,217,216,208
110   DATA  220,215,211,194,206,202,195,203,239,218,219,214,187,210,252,190
120   DATA  196,213,198,199,183,182,249,250,185,252,252,248,247,184,251,253
130   DATA  252,252,252,252,252,224,162,161,252,252,252,252,252,191,188,252
140   DATA  252,252,252,252,252,252,226,225,252,252,252,252,252,252,252,186
150   DATA  228,227,164,165,163,252,229,166,167,252,252,252,252,252,230,252
160   DATA  231,222,223,232,234,233,243,241,240,237,174,173,178,177,176,168
170   DATA  246,254,252,245,244,189,252,252,179,171,242,252,252,252,252,255
180   DIM Pc2hp$[256]
190   CLEAR SCREEN
200   PRINT "Set up PC (Code page 850) to HP (Roman-8) translation string"
210   FOR I=0 TO 255
220     READ C
230     Pc2hp$[I+1;1]=CHR$(C)
240   NEXT I
250   CONFIGURE KBD 0 TO Pc2hp$
260   END
```

See Also:

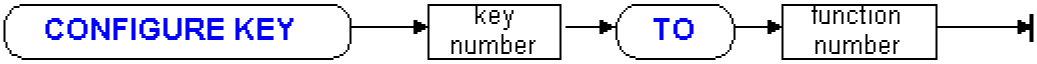
CONFIGURE KEY, CONFIGURE LABEL, LEXICAL ORDER IS

CONFIGURE KEY

Assigns editor functions to keyboard keys.

Syntax: CONFIGURE KEY key-number TO function-number

where: key-number = numeric-expression
function-number = numeric-expression



Sample: `CONFIGURE KEY 1 TO NUM("<")`

View Sample: **CONFIGURE KEY.BAS** **(also found in examples directory)**

Description:

CONFIGURE KEY specifies what keyboard function a keyboard key generates. This statement is version dependent. Statements generated for the DOS version of HTBasic will not work with the Windows version, etc. See "Using the Integrated Environment" in the *Installing and Using* manual for an explanation of how this statement is used in each specific version. The following example makes the Backspace key generate the LEFT function CHR$(255)&"<":

```
CONFIGURE KEY 1 TO NUM("<")
```

See Also:

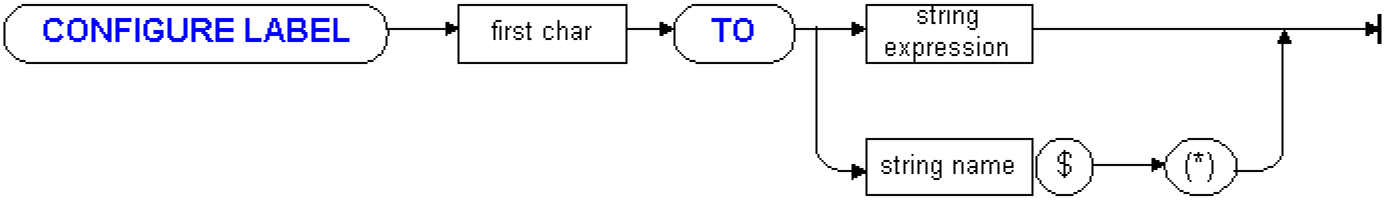
CONFIGURE KBD

CONFIGURE LABEL

Defines characters for the LABEL statement.

Syntax: CONFIGURE LABEL first-char TO string-expression
CONFIGURE LABEL first-char TO string-name$(*)

where: first-char = numeric-expression rounded to an integer.



Sample:

```
CONFIGURE LABEL 128 TO CHR$(128)&CHR$(112)&CHR$(127)&
       CHR$(15)&CHR$(0)
CONFIGURE LABEL 191 TO Newchars$(*)
```

View Sample: **CONFIGURE LABEL.BAS** **(also found in examples directory)**

Description:

CONFIGURE LABEL defines additional characters for use with the LABEL statement. You may define one character by giving a simple string or string expression or several characters by giving a string array. The first-char value specifies the first character to define. Characters in the range 33 to 255 may be defined. To delete a definition, use a zero length string for the definition. See the *User's Guide* for a complete explanation of how to use this feature. Each character in the definition string has the form CHR$(Move + x*16 + y), where Move is 0 or 128, x ranges from 0 (far left) to 7 and y ranges from 0 (bottom) to 15. The baseline is y=5. The following example defines the character "H":

```
CONFIGURE LABEL 72 TO CHR$(133)&CHR$(14)&CHR$(238)&
CHR$(101)&CHR$(138)&CHR$(106)
```

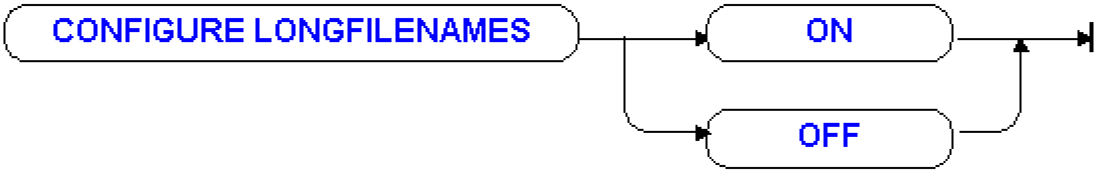
See Also:

CONFIGURE KBD, CONFIGURE KEY, LABEL, LEXICAL ORDER IS

CONFIGURE LONGFILENAMES

Specifies use of long filenames.

Syntax: CONFIGURE LONGFILENAMES { ON | OFF }



Sample:

```
CONFIGURE LONGFILENAMES ON
CONFIGURE LONGFILENAMES OFF
```

View Sample: **CONFIGURE LONGFILENAMES.BAS** **(also found in examples directory)**

Description:

Long filenames are allowed in addition to the standard 8.3 names. The filenames can be 256 characters long and can have embedded spaces. With longfilenames off, HTBasic removes spaces from file specifiers and CAT listings don't have enough room for long filenames. By default use of long filename is enabled. To disable longfilenames use the statement:

```
CONFIGURE LONGFILENAMES OFF
```

With LONGFILENAMES ON, spaces are not deleted from directory and file specifiers since they may be significant. Also, the listing format for CAT is changed to accommodate varying length filenames and four digit years. It is roughly modelled after the NT DIR command listing format.

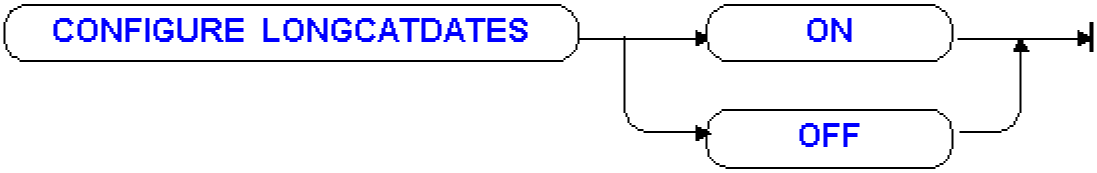
See Also:

CAT

CONFIGURE LONGCATDATES

Specifies use of long dates in CATalogs.

Syntax: CONFIGURE LONGCATDATES { ON | OFF }



Sample:

```
CONFIGURE LONGCATDATES ON
CONFIGURE LONGCATDATES OFF
```

Description:

By default use of long CAT DATES are enabled. To disable long year display use the statement:

```
CONFIGURE LONGCATDATES OFF
```

With LONGCATDATES ON, four digit years are displayed. With LONGCATDATES turned off, the four digit year information is still retained, but is just not displayed.

See Also:

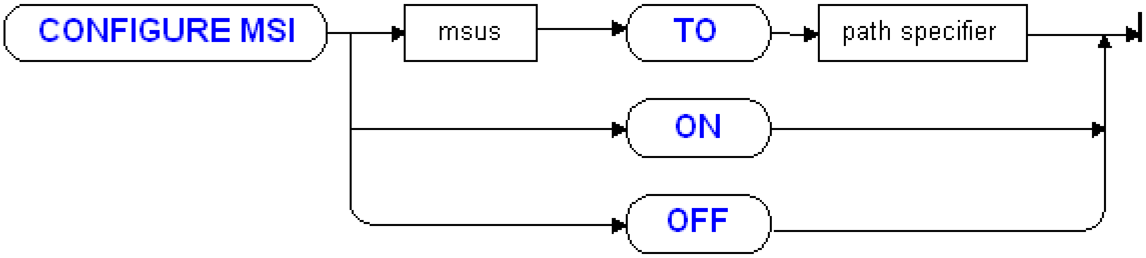
CAT

CONFIGURE MSI

Specifies HP style volume specifier translations.

Syntax: CONFIGURE MSI hp-msus TO path-specifier
CONFIGURE MSI { ON | OFF }

where: hp-msus = string expression of an HP BASIC msus.



Sample:
```
CONFIGURE MSI ":,700,1" TO "c:\TEST\DATA\"
CONFIGURE MSI ":INTERNAL,4,0" TO "/usr/tmp/"
CONFIGURE MSI OFF
```

View Sample: CONFIGURE MSI.BAS (also found in examples directory)

Description:

CONFIGURE MSI specifies a file path-specifier to be substituted for an HP BASIC msus (mass storage unit specifier or volume specifier). Directory names **must** end with a directory separator character. The separator character is the backslash, "\" For example, the following statements would assign the I/O path, @In, to the file "B:\RUN2\DATA":

```
CONFIGURE MSI ":,700,1" TO "B:\RUN2\"
ASSIGN @In TO "DATA:,700,1"
```

Specifying a new path-specifier for a defined hp-msus replaces the previous definition. Specifying a zero length path-specifier removes the previous definition. Note that file names of one letter followed by an hp-msus (i.e., C:,702,1) and file names with an embedded colon (i.e., .xnews.sun:0) will be misinterpreted. **MSI** translation can be turned off with the statement **CONFIGURE MSI OFF** when such conflicts arise. To turn translation back on, use **CONFIGURE MSI ON**.

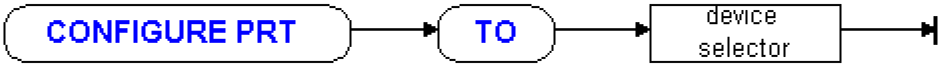
See Also:

MASS STORAGE IS

CONFIGURE PRT

Specifies the value of PRT.

Syntax: CONFIGURE PRT TO device-selector



Sample: `CONFIGURE PRT TO 701`

View Sample: **CONFIGURE PRT.BAS** **(also found in examples directory)**

Description:

CONFIGURE PRT specifies the device-selector that the PRT function returns. It also does an implicit DUMP DEVICE ISPRT. For example, under DOS the following statements output the message "Hello There" to the printer port (assuming 10 is the printer port ISC).

```
CONFIGURE PRT TO 10
OUTPUT PRT; "Hello There"
```

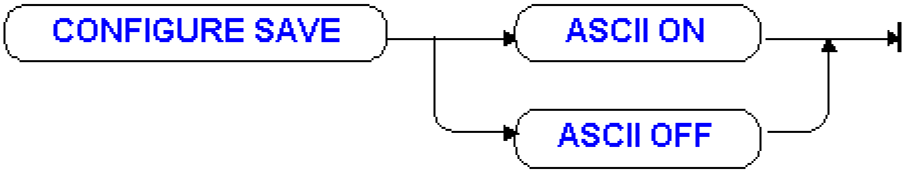
See Also:

DUMP DEVICE IS, PRT

CONFIGURE SAVE

Sets the file type produced by SAVE.

Syntax: CONFIGURE SAVE ASCII { ON | OFF }



Sample: `CONFIGURE SAVE ASCII OFF`

View Sample: CONFIGURE SAVE.BAS **(also found in examples directory)**

Description:

CONFIGURE SAVE ASCII sets the file type SAVE uses when saving a file to disk. **SAVE ASCII ON**, the default, produces a LIF ASCII file. This type of file is useful for exchanging programs with older HP BASIC workstations that can not GET DOS ASCII or UNIX ASCII program files. The *Installing and Using* manual has more information on Diskette Transfer Utilities.

SAVE ASCII OFF produces an ordinary ASCII file. Such a file is compatible with all popular program editors, most word processors and newer releases of HP BASIC. RE-SAVE produces the same file type as an existing file or the file type specified by **CONFIGURE SAVE ASCII** if no file exists. GET can read either file type.

If you use **CONFIGURE SAVE ASCII OFF** you should not embed carriage-returns or line-feeds in string literals since GET will interpret them as end-of-line indicators.

See Also:

CONFIGURE BDAT, CONFIGURE CREATE, RE-SAVE, SAVE

CONFIGURE SYSTEM

Returns the conjugate of a complex number.

Syntax: CONFIGURE SYSTEM (“DEVICE; OPTION”)

Sample:

```
CONFIGURE SYSTEM (“CAT;RECURSIVE”)
CONFIGURE SYSTEM (“CAT”)  !Sets to default
CONFIGURE SYSTEM (“DUMP; PLUS”)
CONFIGURE SYSTEM (“DUMP”) !Sets to default
CONFIGURE SYSTEM (“HPBDAT;READEOF")
CONFIGURE SYSTEM ("HPBDAT")
```

Description:

CONFIGURE SYSTEM sets the system parameters. To set a device to default, omit the option parameter.

Options

CAT RECURSIVE – Using the RECURSIVE option displays file counts during CAT commands.

DUMP PLUS – Normally the DUMP command only copies the contents of the main HTBasic child window. With this option on, all windows inside the HTBasic parent window will be copied to the DEMP device.

"HPBDAT;READEOF" - for proper reading of BDAT files with HP style headers. The default setting is to report an error on EOF rather than to read the contents of the file. The default behavior is to disable reading of the file. This option may be placed back to the default using: CONFIGURE SYSTEM "HPBDAT"

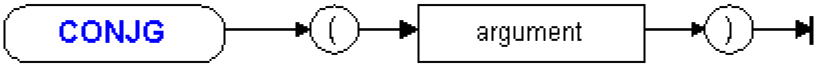
See Also:

CONFIGURE BDAT, CONFIGURE CREATE

CONJG

Returns the conjugate of a complex number.

Syntax: CONJG(numeric-expression)



Sample: `C=SQRT(Z*CONJG(Z))`

View Sample: **CONJG.BAS** **(also found in examples directory)**

Description:

CONJG(Z) is defined as

CONJG(Z) = CMPLX(REAL(Z), -IMAG(Z))

Notice that the real part is unchanged. If the imaginary part is positive, it will be made negative. If the imaginary part is negative, it will be made positive. The effect in the complex domain is to mirror the number about the real axis.

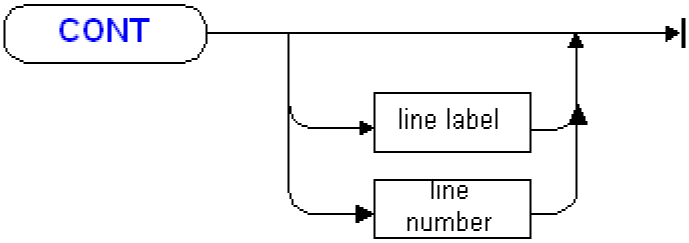
See Also:

ABS, ARG, CMPLX, IMAG, REAL

CONT

Restarts a program which is PAUSEd.

Syntax: CONT [line-number | line-label]



Sample:

```
CONT
CONT 550
CONT Thislabel
```

Description:

A program which is in the Paused state (as indicated by the Run Indicator) can be restarted with the CONTINUE key, button, menu, or with the **CONT** command. If you specify a line number or line label, it must be in the current context or the MAIN context and execution continues at the specified line. If no line is specified, execution resumes at the next line which would have been executed had the program not been PAUSEd.

CONT can be used interactively to debug a program or to restart an un-intentionally aborted program. Variables retain their current values. While the program is PAUSEd, you can see and change the values of variables and use any commands that do not change the program and then **CONT**inue the program. If a change is made to any program statement, the program is stopped and you cannot continue its execution with **CONT**.

This command can only be executed from the keyboard. It cannot be included in a program.

See Also:

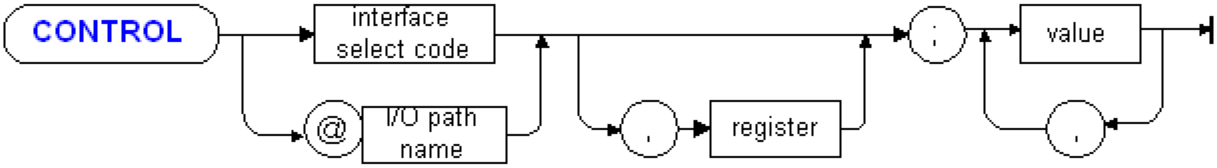
PAUSE, RUNLIGHT

CONTROL

Sends control information to an interface, I/O path, or widget attribute(s).

Syntax: CONTROL dest [,register] ; value [,value...]

where: dest = @io-path | interface-select-code
register = numeric-expression rounded to an integer
value = numeric-expression | numeric-array(*)



Sample:
```
CONTROL @Path,5;Record
CONTROL 2;Column,Line
CONTROL 1801,19;Gains(*)
CONTROL @Strip2;SET (“CURRENT AXIS”:“X”, “RANGE”:20)
CONTROL @Slider;SET (“VALUE”:Setpoint)
```

View Sample: **CONTROL.BAS** **(also found in examples directory)**

Description:

Use **CONTROL** to send control information to an interface or set parameters associated with an I/O
path. Information is sent by specifying a starting register and a value. If no register is specified, register zero is used. If you specify more than one value, the register number is incremented by one
after writing each value.

If the destination is an I/O Path, information is set in the I/O path rather than being sent to the device or file. If the destination is an interface select code (ISC), then the information is sent to the
device driver for interpretation. Consult the documentation for a particular device to find the usage
for each register.

The range of legal registers and the meaning of values written to them differ for each interface. The *User's Guide* describes the **CONTROL** and STATUS registers for many of the interfaces and for I/O paths.

Basic Plus

Each widget has a variety of attributes that control its appearance and behavior. The CONTROL command is used to assign a new value to a widget attribute. The widget must
have been created previously using an ASSIGN statement. Attributes are either scalar (may contain a single value) or vector (may be assigned an array of values) and have values of either numeric or string type.

You can use a shorthand method to assign values to several scalar attributes without naming them individually on the ASSIGN statement. To do this, you store all the attributes
in a string array and all the matching values in another array of the same size.

Then, when you specify both array names in the SET option of the ASSIGN statement, the attribute named in each element of the string array will be assigned the corresponding value in the value array. Elements of the string array that contain nothing, or nothing but
blanks, will be ignored. For example:

```
Attribs$(1) = "X"
Attribs$(2) = "Y"
Attribs$(3) = "WIDTH"
Attribs$(4) = "HEIGHT"
Values(1) = 5
Values(2) = 5>
Values(3) = 500
Values(4) = 300
```

```
CONTROL @Panel;SET (Attribs$(*):Values(*))
```

Porting to HP BASIC:

TransEra has added capabilities to several of the standard interfaces. The additional registers resulting from these enhancements are always numbered 100 and above. In some instances HTBasic can pass arrays to and from a single register. This capability is used for things like gain control lists in data acquisition drivers. These new features are not available in HP BASIC. They should not be used in programs that must be ported back to HP BASIC.

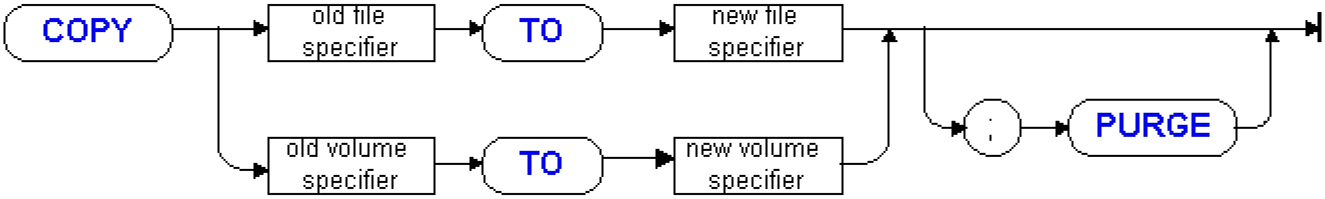
See Also:

STATUS, READIO, WRITEIO

COPY

Copies files.

Syntax: COPY old-file-specifier TO new-file-specifier [;PURGE]



Sample:

```
COPY "Oldfile" TO A$&VAL$(I)
COPY "/Empl1/AFile" TO "/Empl2/AFile"
COPY A$&B$ TO "A:\DIR\FILE";PURGE
```

View Sample: **COPY.BAS** **(also found in examples directory)**

Description:

COPY makes a duplicate copy of a file and gives it a new name. Use the **COPY** command as a program statement or as a keyboard command. If the new-file-specifier already exists, an error is reported if PURGE is not present. If PURGE is present, any existing file named new-file-specifier will be replaced.

HTBasic does not support the copy of a full disk to another disk. Use the operating system for full disk copies. You can use the DOS "DISKCOPY" or "XCOPY" commands. If wildcards are included in the command, then several files can be copied with a single command.

See Also:

CAT, CREATE, CREATE ASCII, CREATE BDAT, MASS STORAGE IS, PERMIT, PROTECT, PURGE, RENAME, SYSTEM$("MSI")

COPYLINES

Copies program lines from one location to another.

Syntax: COPYLINES start [,end] TO target

where: start, end, and target = line-number | line-label

Sample:

```
COPYLINES 10,100 TO 500
COPYLINES 1500 TO 2222
COPYLINES ALabel,BLabel TO Clabel
```

Description:

Use **COPYLINES** to copy a block of lines to a new location, while leaving the original lines untouched. This differs from the MOVELINES statement since the MOVELINES statement deletes the original program portion. If no ending line is specified, only one line is copied. The target line cannot be in the range specified by start and end. If start doesn't exist, the line immediately after that line number is used. If end doesn't exist, the line immediately before that line number is used. If a non-existent line label is specified, an error will be reported.

Line numbers and labels are renumbered and updated if needed. However, line number references in lines not being copied remain linked to the original lines rather than the newly created lines. **COPYLINES** may not copy lines containing a SUB program or DEF FN definition unless the new line number is greater than any existing line number. An error will be issued if this is not the case. This is because a SUB or DEF FN must follow all previous lines. If an error occurs during a **COPYLINES**, the copy is terminated and the program is left partially changed.

This command can only be executed from the keyboard while no program is running. It cannot be included in a program.

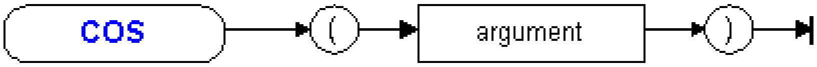
See Also:

CHANGE, DEL, DELSUB, EDIT, FIND, INDENT, MOVELINES, REN, SECURE, XREF

COS

Returns the cosine of an expression.

Syntax: COS(numeric-expression)



Sample:

```
A=COS(B)
Cosine=COS(X+45)
PRINT Cosine+COS(Angle)
```

View Sample: **COS.BAS** **(also found in examples directory)**

Description:

The range of the cosine function is -1 and 1 inclusive. The numeric expression is treated as an angle in the current trigonometric mode: RADians or DEGrees. The default trigonometric mode is RADians.

COMPLEX Arguments

COS accepts either a COMPLEX or REAL argument and returns a value of the same type. For COMPLEX arguments the angle must be specified in radians, regardless of the current trigonometric mode. The real and imaginary parts of **COS**(Z) are calculated (using real arithmetic) as

REAL(COS(Z)) = COS(REAL(Z))*COSH(IMAG(Z))
IMAG(COS(Z)) = -SIN(REAL(Z))*SINH(IMAG(Z))

Notice that intermediate values generated during the calculation of the function can cause over or underflow errors for very large or small values of Z.

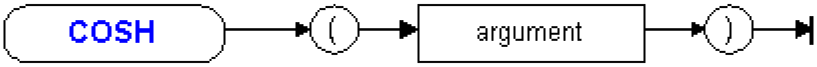
See Also:

ACS, ASN, ATN, SIN, TAN, ASNH, ACSH, ATNH, COSH, SINH, TANH, DEG, PI, RAD

COSH

Returns the hyperbolic cosine of an expression.

Syntax: COSH(numeric-expression)



Sample:

```
A=COSH(B)
Hcosine=COSH(X+PI)
PRINT COSH(CMPLX(X,Y))
```

View Sample: **COSH.BAS** **(also found in examples directory)**

Description:

COSH accepts either a COMPLEX or REAL argument and returns a value of the same type. The argument must be specified in radians, regardless of the current trigonometric mode. The real and imaginary parts of **COSH**(Z) are calculated (using real arithmetic) as

REAL(COSH(Z)) = COSH(REAL(Z))*COS(IMAG(Z))
IMAG(COSH(Z)) = SINH(REAL(Z))*SIN(IMAG(Z))

Notice that intermediate values generated during the calculation of the function can cause over or underflow errors for very large or small values of Z.

See Also:

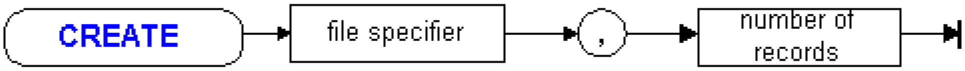
ACSH, ASNH, ATNH, COS, SINH, TANH

CREATE

Creates an ordinary file.

Syntax: CREATE file-specifier,records

where: records = numeric-expression, rounded to an integer.



Sample:

```
CREATE "DOSASCII.TXT",75
CREATE "C:"&Filename$,Size
CREATE "/Net2/Users/Lori/AFile",50
```

Description:

The **CREATE** statement creates an ordinary file of the specified length on the mass storage media, in the specified directory or in the current working directory. **CREATE** does not open files; use ASSIGN to open files. Since Windows supports extendable files, the number of records is ignored and the file is created with a length of zero.

HTBasic supports ordinary files as well as typed files. HTBasic file types are LIF ASCII, BDAT, BIN and PROG. In a CAT listing, the file type column is blank for ordinary files. Unlike typed files, no special header or other embedded information is placed in the file. An ordinary file with FORMAT ON is compatible with all programs that support DOS ASCII files.

Do not confuse the terms ASCII (DOS ASCII, Windows ASCII, UNIX ASCII, etc.) and LIF ASCII. A Windows ASCII file is an ordinary file which contains only printable characters and the end of each line is marked with a carriage return and line feed. A UNIX ASCII file is an ordinary file which contains only printable characters and the end of each line is marked with a line feed. A LIF ASCII file is a typed file which contains string items preceded by an item length and followed by a pad byte when the string length is odd. When the term "ASCII" is used in the HTBasic manual set or in a CAT listing, it refers to LIF ASCII. When the term is used outside the manual set, you will need to determine for yourself what kind of ASCII is spoken of.

See Also:

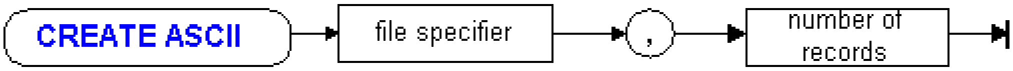
ASSIGN, CAT, COPY, CREATE ASCII, CREATE BDAT, CREATE DIR, MASS STORAGE IS, PURGE, RENAME, PERMIT, PROTECT, SYSTEM$("MSI")

CREATE ASCII

Creates a LIF ASCII file.

Syntax: CREATE ASCII file-specifier,records

where: records = numeric-expression, rounded to an integer



Sample:

```
CREATE ASCII "Tables",75
CREATE ASCII "C:"&Text$,Size
CREATE ASCII "/DirX/DirY/DirZ/AFile",50
```

View Sample: **CREATE ASCII.BAS** **(also found in examples directory)**

Description:

The **CREATE ASCII** statement creates a LIF ASCII file of specified length on the mass storage media, in the specified directory or in the current working directory. **CREATE ASCII** does not open files; use ASSIGN to open files. Since Windows supports extendable files, the file is created with a length of zero, but a CAT listing shows the number of records specified in the CREATE.

HTBasic supports typed files as well as ordinary files. HTBasic file types are LIF ASCII, BDAT, BIN and PROG. In a CAT listing, LIF ASCII files are listed as "ASCII" files.

A utility program, HPCOPY, is provided for most versions of HTBasic to transfer LIF ASCII files between HP LIF diskettes and DOS disks. Data can also be transferred between HTBasic and Series 200/300 computers by attaching an interface between the computers and writing a short program on each computer to transfer the data. Programs can be transferred in ASCII using either of these methods.

Do not confuse the terms ASCII (DOS ASCII, NT ASCII orUNIX ASCII), and LIF ASCII. A Windows ASCII file is an ordinary file which contains only printable characters, and the end of each line is marked with a carriage return and line feed. A UNIX ASCII file is an ordinary file which contains only printable characters and the end of each line is marked with a line feed. A LIF ASCII file is a typed file which contains string items preceded by an item length and followed by a pad byte when the string length is odd. When the term "ASCII" is used in the HTBasic manual set or in a CAT listing, it refers to LIF ASCII. When the term is used outside the manual set, you will need to determine for yourself what kind of ASCII is spoken of.

File Headers

As opposed to ordinary files, typed files have a header containing necessary information about the file. The presence of the header is transparent to BASIC programs and no action should be taken to account for it. HTBasic can work with files that have either an HTB or an HP LIF file header. The HTB file header is 256 bytes. The HP LIF file header is 512 or 768 bytes. The CONFIGURE CREATE statement determines which kind of header is created by this statement. By default, HTB file headers are created.

See Also:

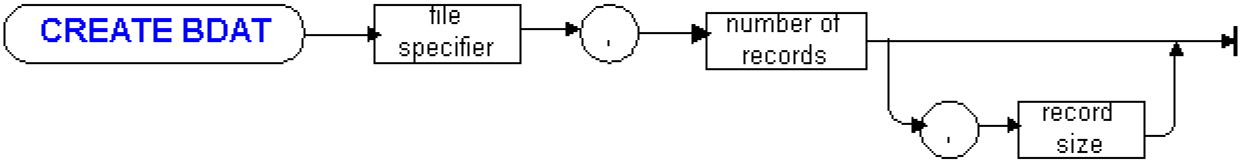
ASSIGN, CAT, COPY, CREATE, CREATE BDAT, CREATE DIR, MASS STORAGE IS, PURGE, RENAME, PERMIT, PROTECT, SYSTEM$("MSI")

CREATE BDAT

Creates a BDAT (binary data) file.

Syntax: CREATE BDAT file-specifier, records [,record-size]

where: records = numeric-expression, rounded to an integer.
record-size = numeric-expression, rounded to integer, then rounded up to even integer or one.



Sample:

```
CREATE BDAT "Doc",50
CREATE BDAT Vol$&Rec$,Bytes,1
CREATE BDAT "/usr/bin/Group",10
```

View Sample: CREATE BDAT.BAS **(also found in examples directory)**

Description:

The **CREATE BDAT** statement creates a binary data file with the specified length and record size on the mass storage media, in the specified directory or in the current working directory. **CREATE BDAT** does not open files; use ASSIGN to open files. Since Windows supports extendable files, the file is created with a zero length, but a CAT listing shows the number of records specified in the CREATE.

The record-size is a numeric expression, rounded to an integer in the range 1 to 65534 and should be an even integer or one. This specifies the number of bytes per record. The default is 256 bytes.

A utility program is provided with most versions of HTBasic to transfer BDAT files between HP LIF diskettes and DOS disks. Data can also be transferred between HTBasic and Series 200/300 computers by attaching an interface between the computers and writing a short program on each computer to transfer the data.

BDAT files must be written with **MSB FIRST** in order for the data to be correctly readable by a Series 200/300 computer. CONFIGURE BDAT can be used to specify the default byte ordering of created files. If CONFIGURE BDAT is not used, HTBasic creates BDAT files using the native byte order of the computer.

File Headers

HTBasic supports typed files as well as ordinary files. HTBasic file types are LIF ASCII, BDAT, BIN and PROG. As opposed to ordinary files, typed files have a header containing necessary information about the file. The presence of the header is transparent to BASIC programs and no action should be taken to account for it. HTBasic can work with files that have either an HTB or an HP LIF file header. The HTB file header is 256 bytes. The HP LIF file header is 512 or 768 bytes. The CONFIGURE CREATE statement determines which kind of header is created by this statement. By default, HTB file headers are created.

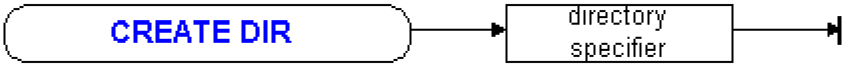
See Also:

ASSIGN, CAT, COPY, CREATE, CREATE ASCII, CREATE DIR, MASS STORAGE IS, PERMIT, PROTECT, PURGE, RENAME, SYSTEM$("MSI")

CREATE DIR

Creates a directory.

Syntax: CREATE DIR directory-specifier



Sample:

```
CREATE DIR "../branch/leaf"
CREATE DIR "C:\ADIR\BDIR"
CREATE DIR "SUB"
```

View Sample: **CREATE DIR.BAS** **(also found in examples directory)**

Description:

CREATE DIR creates a directory and is almost exactly like the HFS or SRM command of the same name. It is the equivalent of the DOS MD or MKDIR commands.

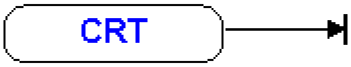
See Also:

CAT, COPY, CREATE, CREATE ASCII, CREATE BDAT, MASS STORAGE IS, PERMIT, PROTECT, PURGE, RENAME, SYSTEM$("MSI")

CRT

Returns the integer 1, the CRT interface select code.

Syntax: CRT



Sample: `PRINTER IS CRT`

```
ENTER CRT;Array$(*)
```

View Sample: CRT.BAS **(also found in examples directory)**

Description:

The **CRT** function always returns the constant 1. It is a useful mnemonic and documentation tool in referring to the **CRT** interface select code.

See Also:

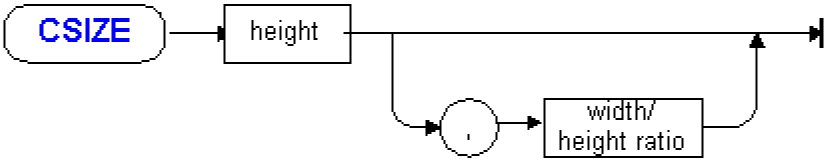
KBD, PRT

CSIZE

Sets the character size for LABEL and SYMBOL.

Syntax: CSIZE height [, expansion-factor]

where: height = numeric-expression
expansion-factor = numeric-expression



Sample: `CSIZE 8`

```
CSIZE 10,0.7
CSIZE Height,Width/Height
```

View Sample: **CSIZE.BAS** **(also found in examples directory)**

Description:

CSIZE sets the character size (height) and the expansion factor (width/height) of the text generated by the LABEL and SYMBOL statements. They are specified in graphic display units. A negative height or expansion-factor inverts the character in relation to that dimension. The default character height is 5 and the default expansion factor is 0.6. These values are in effect at start-up or when GINIT is executed or RESET is pressed.

See Also:

LABEL, LDIR, LORG, SYMBOL

CSUB

Compiled SUBprograms.

Description:

CSUBs are compiled **subprograms** that are created with special tools. **CSUB**s are loaded with LOADSUB and deleted with DELSUB. A **CSUB** looks like a SUB statement and it is called with a CALL statement. A **CSUB** cannot be created or changed in BASIC and therefore any operation that checks for syntax cannot be used. However, operations that are not syntax checked (renumber, etc.) are allowed on a **CSUB**

The HTBasic Numeric Compiler is the primary tool for creating CSUBs. This compiler is designed so the casual HTBasic user can produce fast numerically intensive subprograms. The user writes one or more SUBs in BASIC which contain the calculation intensive code in his program. The SUB or SUBs are then compiled, creating CSUBs which execute many times faster than the original BASIC. No additional programming skill is necessary. Speed of execution of numerically intensive subprograms is the main goal of this compiler.

The CSUB Toolkits allows creation of **CSUB**s in C++. Creation of **CSUB**s with the CSUB Toolkit requires programming experience in C/C++.

See Also:

CALL, DELSUB, LOADSUB, READIO, WRITEIO

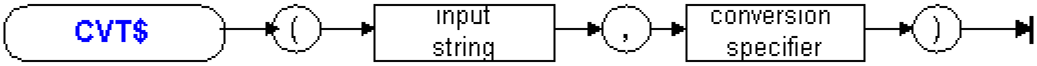
CVT$

Convert strings from one alphabet to another.

Syntax: CVT$(old-string, cvt-name)

where: old-string = string-expression

cvt-name = string-expression



Sample:

```
A$ = CVT$(B$,"HANKAKU KATAKANA TO HANKAKU HIRAGANA")
A$ = CVT$(B$,"HANKAKU HIRAGANA TO HANKAKU KATAKANA")
```

Description:

The **CVT$** string function translates the characters in *old-string* from one alphabet to another. It converts the string character by character and handles a mixture of one- and two-byte character strings.

The CVT$ string function is used for two-byte languages like Japanese. The legal values for *cvt-name*, available alphabets and character mapping between alphabets depends on the specific version of HTBasic.

See Also:

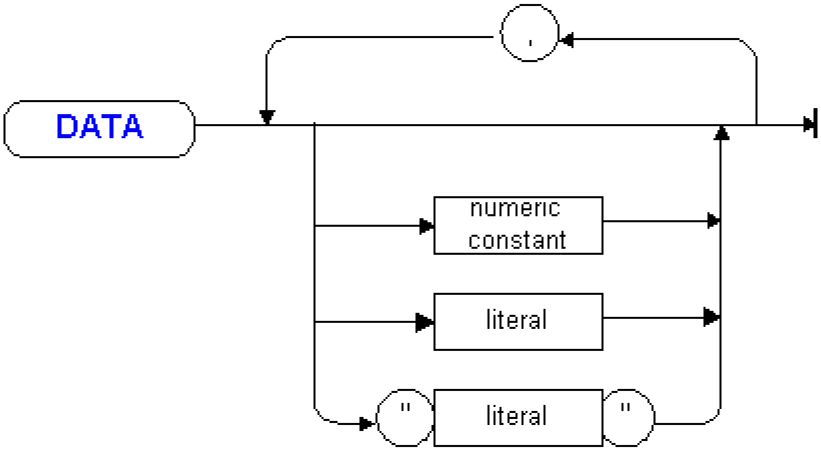
FBYTE, SBYTE

DATA

Stores data items in the program.

Syntax: DATA [data-item] [,data-item...]

where: data-item = ["] string-literal ["] | numeric-constant



Sample:

```
DATA 1.9,"Counts",3.14,56,"Number of Events"
DATA item1,item2,item3
DATA "comment-tail: !","comma: ,","quote: """
DATA 1984,Number of Days
```

View Sample: **DATA.BAS** **(also found in examples directory)**

Description:

DATA and READ statements can quickly and easily provide values for program variables. All **DATA** statements in a context form a single data list. Each context (main program and subprograms) has its own data list. Each variable in the variable list of a READ statement picks up a value from the **DATA** list, starting in sequence: the first variable in a READ picks up the first value in the data list, then the next variable picks up the next value, etc. When a subprogram is called, the current point in the sequence is remembered and restored when control returns to the calling context.

The **DATA** items are treated as literals making it necessary for the computer to process the numeric variables with the VAL function. An error is generated if string values are found in numeric variables, but numeric values may be placed in string variables. Leading and trailing blanks are deleted from unquoted literals. Unquoted literals cannot contain quote marks, comment tails or commas. To include one of these characters in a literal, you must use quotation marks around the literal. A quotation mark is included inside the literal by using two quote marks in the place where you wish to have one. To include a COMPLEX number in a **DATA** statement, list the real and imaginary parts separately, separated by a comma.

You can make a READ start at the beginning of any **DATA** statement by using a RESTORE command.

See Also:

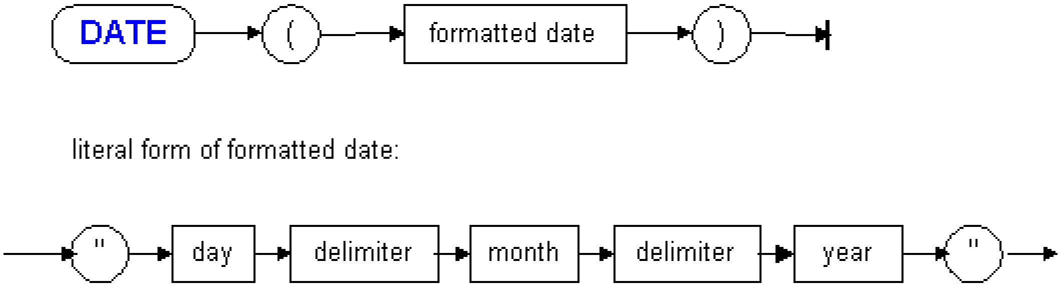
READ, RESTORE

DATE

Converts a string representing a date to a number of seconds.

Syntax: DATE(date-string)

where: date-string = string-expression.



Sample:

```
SET TIMEDATE DATE("6 NOV 1992")
Cycle=DATE("7 JAN 1988)-DATE("1 JAN 1988")
```

View Sample: **DATE.BAS** **(also found in examples directory)**

Description:

The date, encoded in a string in the form "DD MMM YYYY" or "DD MONTH YYYY", is converted to the number of seconds since the start of the Julian Period in 4713 BC.

If **DATE** is used as the argument for SET TIMEDATE, then the clock will be set to midnight of the date specified in the **DATE** argument. The date must be within the legal range supported by your operating system.

Actually, the Rocky Mountain BASIC time base is slightly different than the Julian Period, but can easily be converted. The following function converts a date in the form "DD MMM YYYY" to the Julian Day:

```
10  DEF FNJd(A$)
20    RETURN (DATE(A$) DIV 86400)-1
30  FNEND
```

See Also:

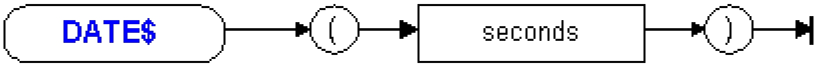
DATE$, SET TIME, SET TIMEDATE, TIME, TIME$, TIMEDATE

DATE$

Takes a numeric value representing seconds and formats it into a date string.

Syntax: DATE$(seconds)

where: seconds = numeric expression.



Sample: `PRINT DATE$(TIMEDATE)`

```
A$=DATE$(Newtime)
```

View Sample: **DATE$.BAS** **(also found in examples directory)**

Description:

If TIMEDATE is used as the argument, **DATE$** returns the current date as a string in the form DD MMM YYYY, where DD is the current day, MMM is the current month in three letter abbreviated form and YYYY is the current year.

The numeric value specified is loosely based on the Julian Period. To convert a Julian Day number to the string form "DD MMM YYYY", use the following function:

```
10  FNJd2date$(Jd)
20    RETURN DATE$((Jd+10)*86400)
30  FNEND
```

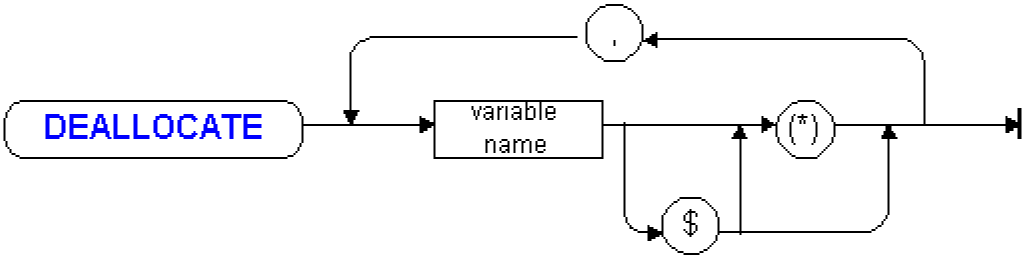
See Also:

DATE, SET TIME, SET TIMEDATE, TIME, TIME$, TIMEDATE

DEALLOCATE

Frees memory space reserved by the ALLOCATE statement.

Syntax: DEALLOCATE variable-name [$] [(*)] [, ...]



Sample: `DEALLOCATE P1$,Aarray(*),Code$(*)`

View Sample: **DEALLOCATE.BAS** **(also found in examples directory)**

Description:

ALLOCATE and ON event statements reserve memory on the BASIC stack; therefore, a **DEALLOCATE** request may not immediately free memory for another use if it is not the next area of memory to come off the stack. Subprogram variables, including those ALLOCATEd, are automatically **DEALLOCATE**d upon subprogram exit. If you try to **DEALLOCATE** a variable which is not currently ALLOCATEd, you get an error.

See Also:

ALLOCATE, COM, COMPLEX, DIM, INTEGER, OPTION BASE, REAL, REDIM

DEF FN

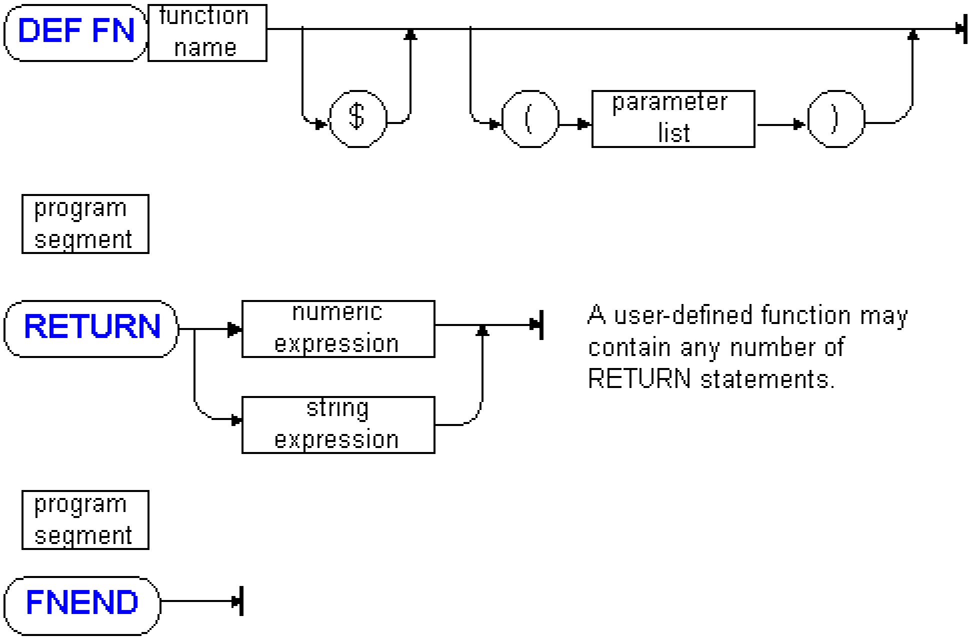
Begins a user-defined function subprogram.

Syntax:

DEF FN function-name[$] [(parameter-list)]
statements
RETURN { numeric-expression | string-expression }
statements
FNEND

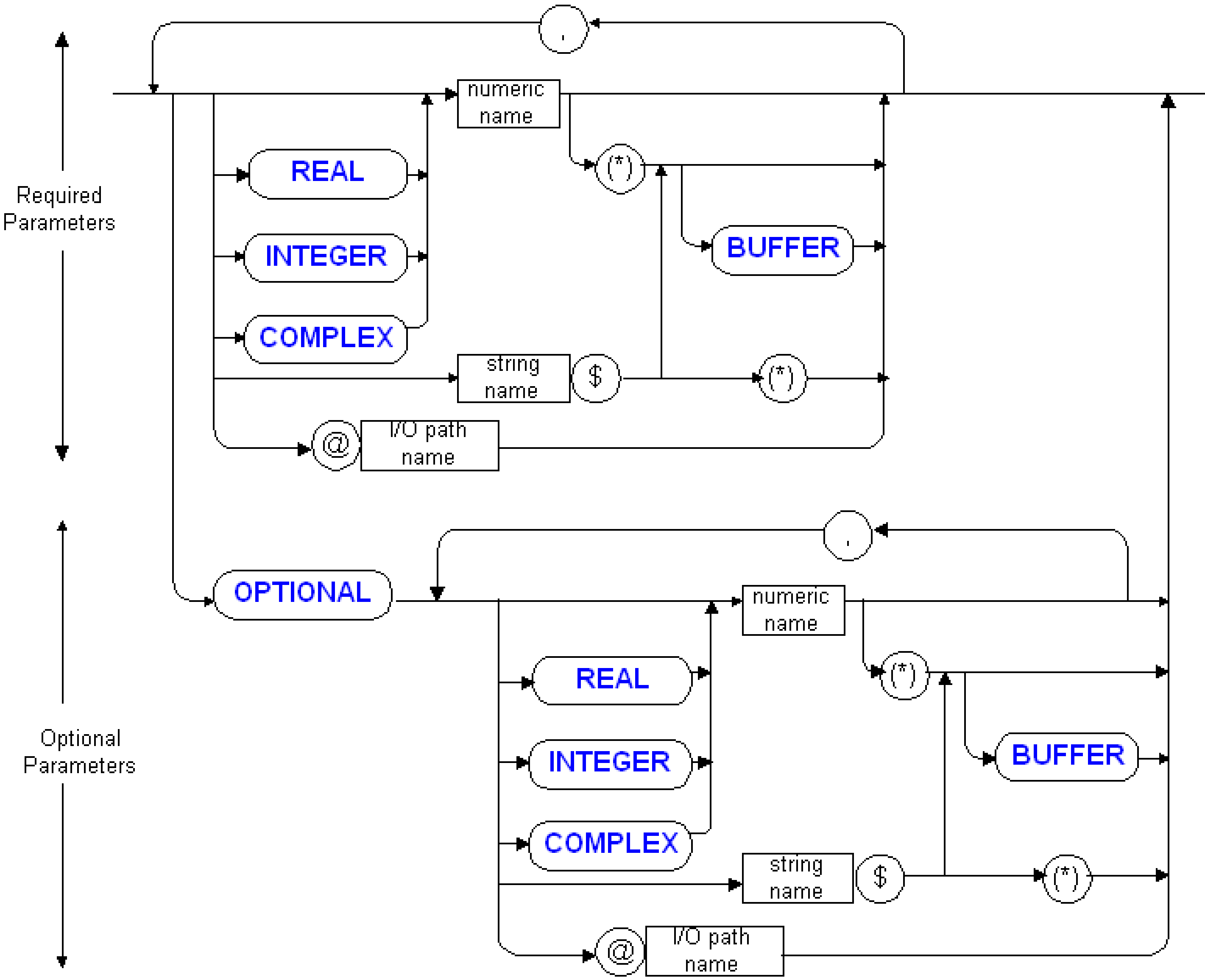
where:

statements = zero, one or more program statements,
including additional RETURN statements.
parameter-list = [param [,param...]] [,] [OPTIONAL param [,param...]]
[,] = the optional comma is only needed when items
occur on both sides of it.
param = [type] numeric-name [(*) [BUFFER]] |
string-name$ [(*) | BUFFER] | @io-path
type = REAL | INTEGER | COMPLEX



A user-defined function may contain any number of RETURN statements.

parameter list:



Sample:

```
DEF FNString$(@Path,REAL Array(*),OPTIONAL Factor$)
DEF FNNum(OPTIONAL X(*))
100  DEF FNFactorial(F)
110    IF F<0 THEN CAUSE ERROR 19
120    IF F<=1 THEN RETURN 1
130    RETURN F*FNFactorial(F-1)
140  FNEND
```

View Sample: DEFFN.BAS **(also found in examples directory)**

Description:

When typing in a new user-defined function subprogram, the **DEF FN** must be the highest numbered line in the present program. The body of the function then follows. SUB or **DEF FN** statements are not allowed inside the body of the function. Lastly, the function definition is completed by a **FNEND** statement. Optionally, comments about the function can follow the **FNEND** statement. At least one **RETURN** statement must exist in the function definition. The **RETURN** statement specifies the value that is to be returned. The type of the value must match the type of the function name; a string function must return a string value and a numeric function must return a numeric value. If execution reaches the **FNEND** statement, an error will result.

When called, a list of arguments can be passed to the function and are associated with the **DEF FN**parameters. Parameters to the right of the OPTIONAL   keyword are optional and need not be passed in the argument list. An error results if the function attempts to use an optional parameter with no value passed to it. To avoid this, use NPAR to check the number of arguments passed to the function.

All variables defined in a subprogram that are not COM variables are local to the subprogram. Upon each entry to the subprogram they are set to zero.

A parameter may be used as a buffer if declared as a BUFFER in both the calling context argument list and the **DEF FN** parameter list. The variables of a parameter list cannot be declared in COM or other variable declaration statements.

Porting Issues

Nested I/O does not return an error under HTBasic but should not be used because future improvements may make it illegal. Using nested I/O also prevents the program from running under HP BASIC.

HTBasic limits the depth that recursion can occur. The depth is limited by the size of the processor stack, not the BASIC workspace size.

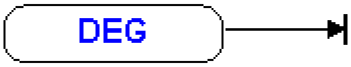
See Also:

CALL, FN, NPAR, SUB

DEG

Sets the trigonometric mode to degrees for all angle measurements.

Syntax: DEG



Sample: `DEG`

View Sample: **DEG.BAS** **(also found in examples directory)**

Description:

All angle arguments and functions that return an angle measurement use the current trigonometric mode which can be either radians or degrees. **DEG** sets the trigonometric mode to degrees. The default trigonometric mode at start-up or after a SCRATCH A is radians.

A subprogram will use the same trigonometric mode as its caller unless it executes a RAD or **DEG** statement. Upon returning to the caller the previous trigonometric mode is restored.

See Also:

ACS, ASN, ATN, COS, RAD, SIN, TAN

DEL

Deletes program lines.

Syntax: DEL start [, end]

where: start and end = line-number | line-label

Sample:

```
DEL 100
DEL Go,Stop
DEL Thislabel,1500
DEL 100,1000
```

Description:

A range of program lines can be deleted by separating the starting and ending line numbers with a comma. If only one line is specified, only that line is deleted. Once a **DEL** statement has been executed, the specified lines cannot be retrieved.

SUB and DEF FN statements can not be deleted unless the entire subprogram is included in the range.

DEL cannot be executed from a running program, but can be executed while the program is PAUSEd (after **DEL** executes, the program is placed in a STOP state).

See Also:

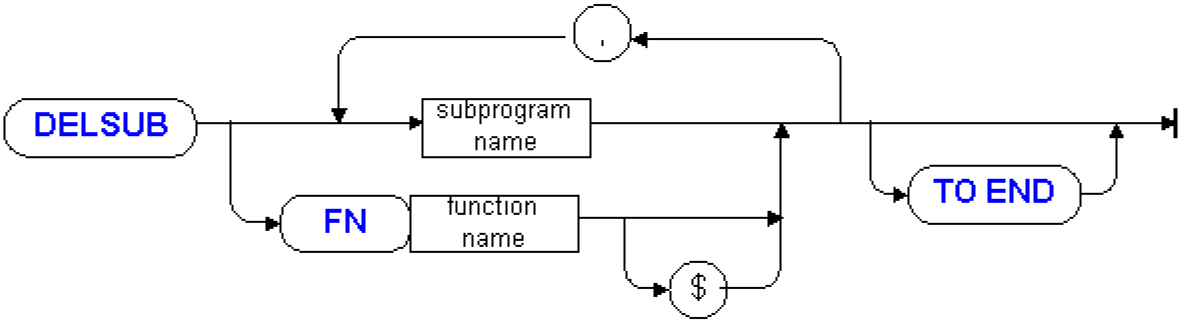
CHANGE, COPYLINES, DELSUB, EDIT, FIND, INDENT, MOVELINES, REN, SECURE, XREF

DELSUB

Deletes SUB or CSUB subprograms from memory.

Syntax: DELSUB context [,context...] [TO END]

where: context = subprogram-name | FN function-name | string-expression



Sample:

```
DELSUB FNProc$
DELSUB Transform TO END
DELSUB Unit1,Unit2,Unit3,Unit4
```

View Sample: **DELSUB.BAS** **(also found in examples directory)**

Description:

DELSUB can delete one or more subprograms, CSUBs, or user-defined function subprograms from memory. If **TO END** is specified in the **DELSUB** statement, then the specified subprogram plus all following subprograms are deleted to the end of the program. If you specify a name and two subprograms both have that name, the first one is deleted. You cannot delete a subprogram if it is currently active or if it is referenced by a currently active ON event statement.

If a string expression specifies the subprogram name in the **DELSUB** statement, the string expression is called a subprogram pointer because it "points" to the subprogram rather than explicitly naming it. As the expression changes, the pointer points to different subprograms. The following example illustrates how this can be useful.

```
10   SUB Xform(X(*))
20     Method$="Xform"&VAL$(RANK(X))
30     IF NOT INMEM(Method$) THEN LOADSUB Method$
40     CALL Method$ WITH(X(*))
50     DELSUB Method$
60   SUBEND
```

The subprogram must be specified with the initial character in uppercase, and subsequent characters in lowercase. Subprogram pointers can also be used in CALL, INMEM, LOADSUB, and XREF statements.

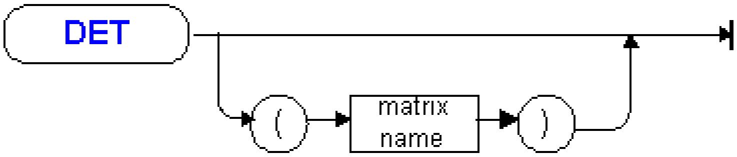
See Also:

CALL, COPYLINES, CSUB, DEF FN, DEL, EDIT, FIND, INMEM, LOADSUB, MOVELINES, REN, SECURE, SUB, XREF

DET

Returns the determinant of a matrix.

Syntax: DET [(numeric-array)]



Sample:

```
Fmatrix=DET
PRINT DET(Fmatrix)
```

View Sample: **DET.BAS** **(also found in examples directory)**

Description:

Use the **DET** function to find the determinant of a matrix. If no argument is given, **DET** returns the determinant of the most recently inverted matrix. Zero is returned if no matrix has been inverted since start-up, SCRATCH or SCRATCH A. If the determinant of a matrix is zero, the matrix does not have a valid inverse. If a very small value is returned compared to the matrix elements, this may imply the matrix cannot accurately be inverted by computer methods.

See Also:

DOT, MAT, SUM

DIALOG

Generates an HTBasic Plus dialog of the specified type.

Syntax: DIALOG

Sample:

```
DIALOG “WARNING”, “Reactor Meltdown Imminent”,Btn;SET (“BACKGROUND”:2)
DIALOG “STRING”, “Enter your Operator ID:”;
RETURN (“VALUE”:Resp$),TIMEOUT 10


DIM Speeds$(0:2)[20]
!
Speeds$(0)= “Fast”
Speeds$(1)= “Slow”
!
DIALOG “LIST”, “Pick your speed:”, Btn;
SET (“WIDTH”:250, “HEIGHT”:80, “ITEMS”:Speeds$(*)),
RETURN (“SELECTION”:Resp)
```

Description:

One of the fundamental HTBasic Plus entities. A dialog is created on the computer screen with the DIALOG statement from an executing BASIC program or from the command line.

The DIALOG statement is a shortcut method for requesting input from the operator. The DIALOG statement functions in a similar fashion to the INPUT and LINPUT statements by collecting operator input without using more complex statements.

Using the DIALOG statement, you can perform the functional equivalent of the following (lengthier) process that would otherwise require more statements.

1. You create a PANEL widget that contains a prompt string, a single widget, and some button widgets.

2. You interact with the contained widget.

3. Then, when you “press” one of the buttons, the system destroys all of the widgets that make up the “dialog PANEL” after passing the selected values from each of the widgets into the variables you have specified.

Types of Dialogs

The types of dialogs that may be created with the DIALOG statement are: COMBO, ERROR, FILE, INFORMATION, KEYPAD, LIST, NUMBER, QUESTION, STRING, and WARNING.

DIALOG/DEFAULT BUTTONS

Use the DIALOG BUTTONS attribute to create the buttons in the dialog. These buttons appear in a single row at the bottom of the dialog, in the same order (left to right on the screen) in which they appear in the attribute array.

To specify one of these buttons as the default button, use the DEFAULT BUTTON attribute. Both DIALOG BUTTONS and DEFAULT BUTTON have different default values, depending on the type of dialog created.

selected button Option

If you specify the optional variable for *selected button*, when the DIALOG statement completes, the variable will contain: an index into the DIALOG BUTTONS array that identifies which button the user pushed to terminate the dialog, or a –1, indicating a timeout has occurred.

The DIALOG BUTTONS array is always treated as OPTION BASE 0, regardless of how it was dimensioned.

SET Option

The SET option is used to specify the initial values for the attributes and to specify the initial values to be displayed by the contained widget (for example, the contents of the STRING widget in the STRING Dialog).

RETURN Option

The RETURN option is used to specify the variables that will receive the final values of the dialog attributes just before the dialog is destroyed. These variables are used primarily to communicate the state of the contained widget back to the program when the user terminates the dialog. For example, to find out what the user typed into a STRING dialog, you should RETURN the VALUE attribute to a string variable in your program.

Specifying the same variable for the same attribute in both the SET and RETURN attribute lists is acceptable, and in fact will be a common practice when the dialog's purpose is to allow the user to modify an existing quantity. For this purpose, you should supply the existing value in the SET attribute list and use the same variable in the RETURN attribute list so the user's modification will change the program variable.

TIMEOUT Option

If you specify the TIMEOUT option in the DIALOG statement, the program will wait only the specified number of seconds for user input before continuing. If the user does not push a button in the dialog within the allotted time, the DIALOG statement will:

- Copy the current state of the dialog attributes into the variables specified in *return attribute list*.

- Destroy the dialog.

- Return a *selected button* value of –1, if the optional variable that will receive this value has been specified.

See Also:

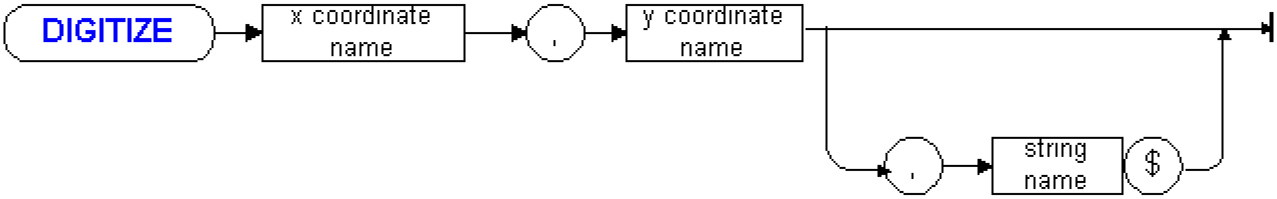
OPTION BASE

DIGITIZE

Inputs digitized X and Y coordinates.

Syntax: DIGITIZE x, y [, string-name$]

where: x and y = numeric-name



Sample:

```
DIGITIZE Xcoor,Ycoor,Stat$
IF Ready THEN DIGITIZE X,Y
```

View Sample: **DIGITIZE.BAS** **(also found in examples directory)**

Description:

A point is digitized from the GRAPHICS INPUT IS device and the coordinates of the point are assigned to the variables. The coordinates are in default units or the units defined in a WINDOW or SHOW statement. A **DIGITIZE** may be completed on the keyboard (if GRAPHICS INPUT IS is from the keyboard) by pressing CONTINUE or ENTER. The **DIGITIZE** statement may optionally specify a status string variable. This 8 byte status variable inputs the status of the GRAPHICS INPUT IS device. The 8 byte status string variable is defined as follows:

Byte	**Meaning**
1	Indicates End of Stream for a device supporting continuous point stream digitizing. Byte 1 may be used as the pen control value in a PLOT. It is "0" if it is the last of a continuous point stream. It is "1" otherwise, including points from a device supporting only single point digitizing.
2	Comma delimiter character.
3	Clip Indicator - If the character is a "0", then the point is outside the hard-clip limits. If a "1", the point is inside the hard-clip limits, but outside the soft-clip limits (see CLIP). If a "2" then it is inside the soft-clip limits.
4	Comma delimiter character.
5	Tracking ON/OFF - If the character is a "0", then tracking is off; if a "1", then tracking is on.
6	Comma delimiter character.
7-8	Button Positions. If S$ is the status string and B is the button number you wish to test, then BIT(VAL(S$[7,8]),B-1) returns one if B is down and zero if B is up.

See Also:

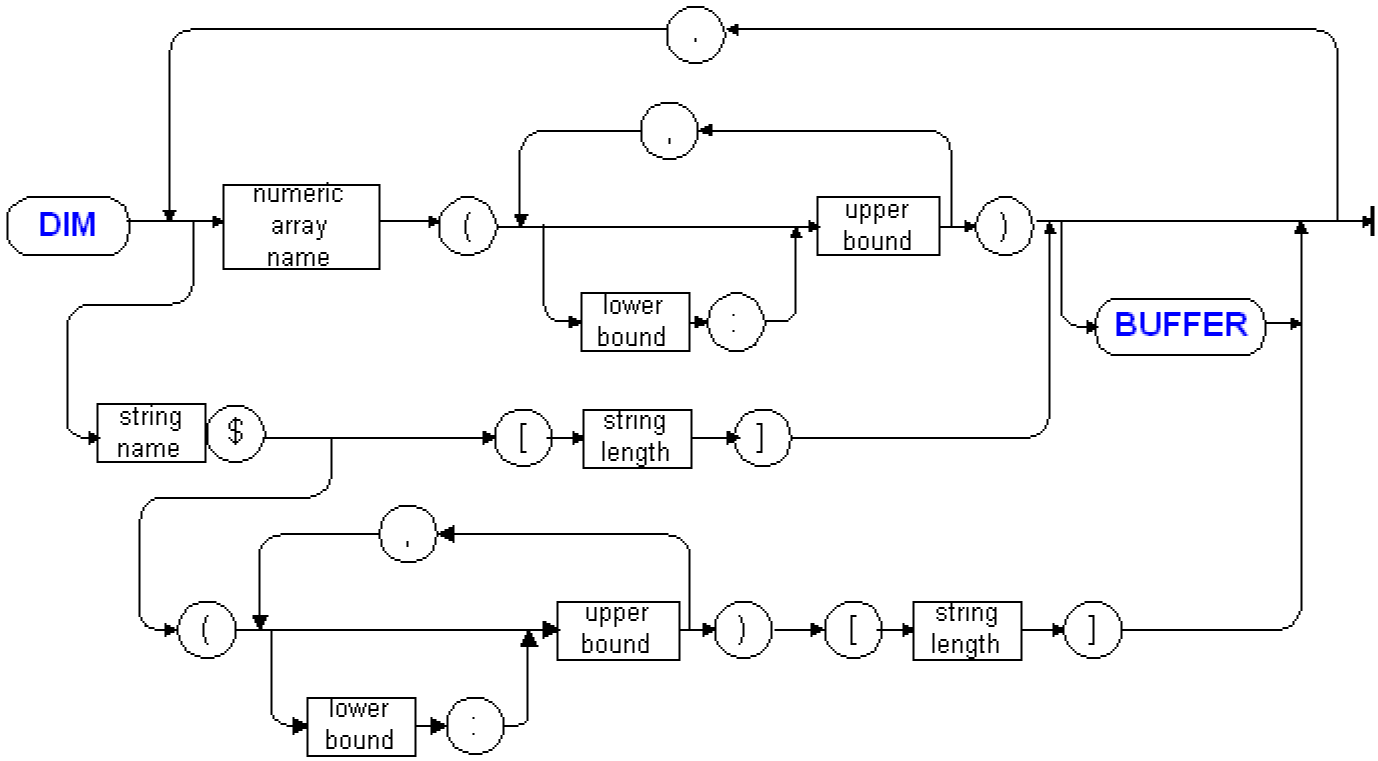
GRAPHICS INPUT IS, READ LOCATOR, TRACK, WHERE

DIM

Dimensions REAL arrays and strings.

Syntax: DIM item [,item...]

where:
item = numeric-name (bounds) [BUFFER] |
string-name$ '['length']' [BUFFER] |
string-name$ (bounds) '['length']'
bounds = [lower-bound :] upper-bound [,bounds...]
bound and length = integer constants



Sample:
```
DIM A(100),B(10,10),C(4,2,5,8)
DIM A$[200],B$(6,10)[100]
DIM Array(-64:63,8)
DIM Hold$[365] BUFFER, Array(200) BUFFER
```

View Sample: **DIM.BAS** **(also found in examples directory)**

Description:

The **DIM** statement is used to declare REAL numeric array and string variables. The maximum number of array dimensions is six and the lower bound must be less than or equal to the upper bound value. Each dimension may contain a maximum of 32,767 elements. The default dimension of an undeclared array is the number of subscripts found in its first occurrence, with each dimension having the default lower bound of the value declared in OPTION BASE and an upper bound of ten.

Each numeric array element is REAL and requires eight bytes of storage. Strings require one byte of storage per character, plus two additional bytes. To declare a variable a BUFFER, follow its name with the BUFFER keyword. BUFFER variables are used with the TRANSFER statement.

Any number of **DIM** statements are allowed, anywhere in the program; however, a **DIM** statement may not appear before an OPTION BASE statement. Memory allocation is made during prerun and cannot be dynamically deallocated. However, the dimensions can be changed in a limited way by REDIM. Use ALLOCATE and DEALLOCATE for dynamic memory allocation.

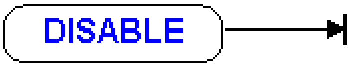
See Also:

ALLOCATE, COM, COMPLEX, DEALLOCATE, INTEGER, OPTION BASE, REAL, REDIM, TRANSFER

DISABLE

Disables event-initiated branches.

Syntax: DISABLE



Sample: `DISABLE`

View Sample: **DISABLE.BAS** **(also found in examples directory)**

Description:

Disables all event-initiated branches, except ON END, ON ERROR, and ON TIMEOUT.

See Also:

DISABLE INTR, ENABLE, ENABLE INTR, ON, OFF

DISABLE EVENT

Prevents HTBasic from branching upon receipt of a specified event.

Syntax: DISABLE EVENT @Eventname,”Widgethandle”

Sample:

```
DISABLE EVENT @Myscrollbar, “Changed”
DISABLE EVENT @String1, “KEYSTROKE”
```

Description:

Use the DISABLE EVENT keyword to temporarily suspend the effects an event would have on your program. For example, you may want the program to accomplish some task without interruption. After that task is accomplished, you can use the ENABLE EVENT keyword to re-enable the program’s sensitivity to that event.

While the event is disabled, it can still be logged. Then, when it is re-enabled the branch will be taken. Only one occurrence of the event will be logged.

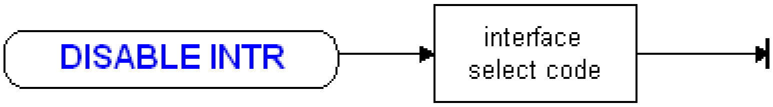
See Also:

DISABLE, DISABLE INTR, ENABLE, ENABLE EVENT, ENABLE INTR

DISABLE INTR

Disables interrupts from the specified interface.

Syntax: DISABLE INTR interface-select-code



Sample: `DISABLE INTR 5`
`DISABLE INTR Isc`

View Sample: **DISABLE INTR.BAS** **(also found in examples directory)**

Description:

DISABLE INTR instructs the interface to disable interrupt generation.

See Also:

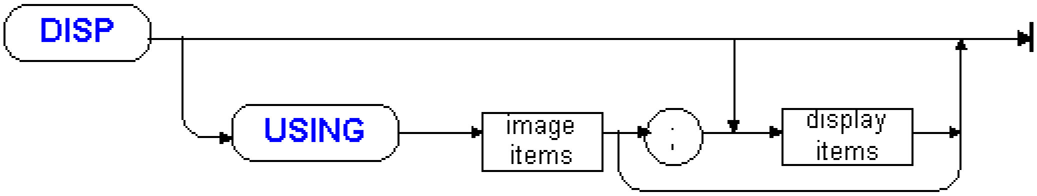
DISABLE, ENABLE, ENABLE INTR, ON INTR, OFF INTR

DISP

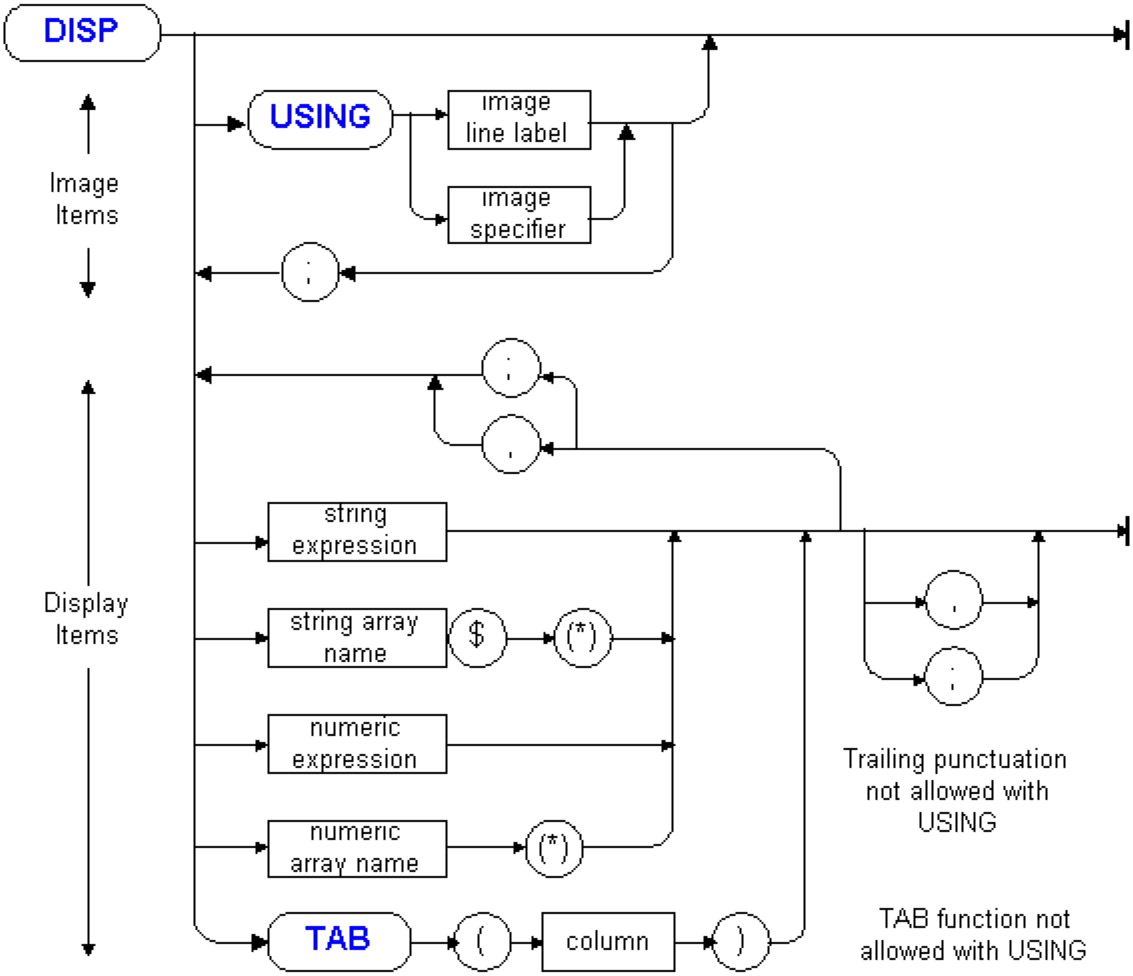
Displays items on the CRT display line.

Syntax: DISP [item-list [{,|;}]]
DISP USING image [; item-list]

where: item-list = item [{,|;}item-list]
item = numeric-expression | numeric-array(*) |
string-expression | string-array$(*) | TAB(column)
column = numeric-expression rounded to an integer
image = line-number | line-label | string-expression
See IMAGE for image syntax.



expanded diagram:



Sample: `DISP Display$;`

```
DISP TAB(8),Head,TAB(25),Descrip
DISP USING "5Z.DD";Figures
DISP USING Report;List(2),List(3),List(4)
```

View Sample: DISP.BAS **(also found in examples directory)**

Description:

Without USING

If **USING** is not specified, the standard numeric format will be used to display items. The standard numeric format will display a number in floating point form rounded to 12 digits if its absolute value is in the range 1E-4 to 1E+6. The number will be displayed in scientific notation if it is outside this range.

The punctuation following the item to be displayed determines the item's display field. The compact field is used if a semicolon follows the item; and the default display field is used if a comma follows the item.

In both compact and default display form, numbers are displayed with one leading blank for positive numbers or the minus sign for negative numbers. In compact field form numeric items are displayed with one trailing blank and string items are displayed with no leading or trailing blanks. The default display form displays items with trailing blanks to fill to the beginning of the next 10-character field. A complex number is displayed in rectangular form, first the real part, then an extra blank and finally the imaginary part.

An array may be displayed in row-major order using the full-array-specifier. If punctuation follows an array then the array elements are displayed either in compact field (if semicolon) or default display field (if comma) and additionally the automatic EOL sequence will be suppressed.

With USING

See IMAGE for a complete explanation of the image list. The items specified in the image list are acted upon as they are encountered. Each image list item should have a matching display item. Processing of the image list stops when no matching display item is found. Conversely, the image list is re-used starting at the beginning to provide matches for all remaining display items. The **TAB** function and any trailing punctuation may not be specified with USING.

Control Characters

The following control characters have a special meaning when used in **DISP** statements:

Character	Meaning
CTRL-G, CHR$(7)	sounds the bell.
CTRL-H, CHR$(8)	moves the cursor back 1 space.
CTRL-L, CHR$(12)	clears the display line (form feed).
CTRL-M, CHR$(13)	moves the cursor to column 1 and the display line is cleared by the next character sent to the display (unless it is a CR).

Scrolling

If the data displayed on the **DISP** line is too long, the data is scrolled to the left so that the final portion is completely displayed. If the **DISP** statement ends with a comma or semicolon, the next **DISP** statement concatenates data on the end of the existing data. Again, the data is scrolled if necessary to display the final portion of the data.

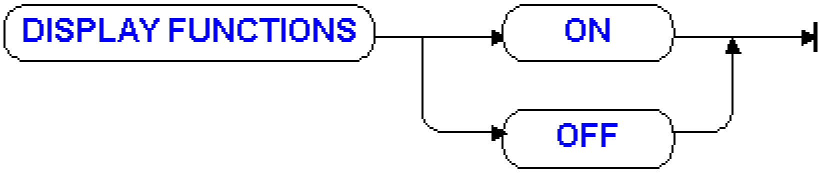
See Also:

IMAGE, LABEL, OUTPUT, PRINT

DISPLAY FUNCTIONS

Controls the display of control characters on the CRT.

Syntax: DISPLAY FUNCTIONS { ON | OFF }



Sample:

```
DISPLAY FUNCTIONS OFF
IF Ctrlchar THEN DISPLAY FUNCTIONS ON
```

View Sample: **DISPLAY FUNCTIONS.BAS** **(also found in examples directory)**

Description:

It is possible to disable the effect of the attribute characters on the CRT device, displaying them instead of executing them. This is useful when debugging OUTPUT. The **DISPLAY FUNCTIONS ON** statement causes all control characters to be displayed but not executed. The only exception is carriage return, CHR$(13), which is first displayed and then the print cursor is moved to column one of the next line. **DISPLAY FUNCTIONS OFF** returns execution of attribute characters to normal.

This function is the equivalent to pressing the DISPLAY FCTNS key or to executing the command, CONTROL CRT,4;State.

See Also:

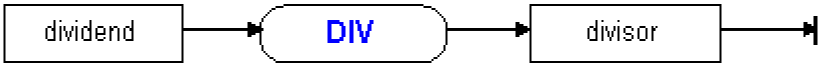
ALPHA HEIGHT, ALPHA PEN, CLEAR LINE, CLS, KBD CMODE, KEY LABELS

DIV

Returns the quotient of an integer divide operation.

Syntax: dividend DIV divisor

where: dividend and divisor = numeric-expressions



Sample: `PRINT "Miles =";Feet DIV 5280`

View Sample: **DIV.BAS** **(also found in examples directory)**

Description:

The result of **DIV** is an INTEGER if both arguments are INTEGER and REAL otherwise. If the divisor is zero, an error is returned. The definition of A **DIV** B is

A **DIV** B = FIX(A/B).

See Also:

MOD, MODULO

DLL GET

Sets up a Dynamic Link Library (DLL) function to use in the program.

Syntax:

DLL GET “*returntype dllname:functionname*” AS “*alias*”

where:

returntype is one of the following: VOID, SHORT, LONG, DOUBLE, CHAR, CHARPTR, VARIABLE.

dllname must be the name of a loaded DLL.

functionname is the name of the function in the DLL you wish to call, or a variable exported from the DLL.

All Function/Variable names must use valid HTBasic function name conventions or an alias using HTBasic function name conventions must be provided. The DLL loader will allow you to load two functions with the same name as long as they are in different DLL’s. However, without an alias specified, there is no way to differentiate which DLL you are trying to call and the DLL loader will always call the first function by that name. You cannot have an HTBasic function with the same name as a DLL function.

alias is an optional function/variable name to use within HTBasic.

Sample:

```
DLL GET “VOID Pipecalc:Xsection” AS “Cross”
DLL GET “SHORT Pipecalc:Xsection” AS “Cross”
DLL GET “LONG Pipecalc:Xsection” AS “Cross”
DLL GET “DOUBLE Pipecalc:Xsection” AS “Cross”
DLL GET “CHAR Pipecalc:Xsection” AS “Cross”
DLL GET “CHARPTR Pipecalc:Xsection” AS “Cross”
DLL GET “VARIABLE Pipecalc:Xsection” AS “Cross”
```

Description:

The DLL GET sets up a Dynamic Link Library (DLL) function to use in the program.

See Also:

DLL LOAD, DLL READ, DLL UNLOAD, DLL WRITE, LIST DLL

DLL LOAD

Specifies the Dynamic Link Library (DLL)   to LOAD into the program.

Syntax:

DLL LOAD “dllname”

where:

dllname must be the name of a DLL to load.

Sample:

```
DLL LOAD “Pipecalc”
DLL LOAD “Flowtrak”
```

Description:

The DLL LOAD specifies the Dynamic Link Library (DLL)   to LOAD into the program.

See Also:

DLL GET, DLL READ, DLL UNLOAD, DLL WRITE, LIST DLL

DLL READ

Retrieves a Dynamic Link Library (DLL) variable to use in the program.

Syntax:

DLL READ “*varname*”;*basic variable*

where:

varname is any variable name within the DLL.

basic variable is any legal variable name to use within HTBasic.

Sample:

```
DLL READ ”Xsection”;Crosec
DLL READ ”Flowrate”;Torrant
```

Description:

The DLL READ reads a loaded Dynamic Link Library (DLL) variable value into a BASIC variable.

See Also:

DLL GET, DLL LOAD, DLL UNLOAD, DLL WRITE, LIST DLL

DLL UNLOAD

Specifies the Dynamic Link Library (DLL) to UNLOAD from the program.

Syntax:

DLL UNLOAD “dllname”
or
DLL UNLOAD ALL

where:

dllname must be the name of a DLL to unload.

Sample:

```
DLL UNLOAD ALL         !Removes all loaded DLLs
DLL UNLOAD “Flowtrak” !Removes Flowtrack.dll
```

Description:

The DLL UNLOAD specifies the Dynamic Link Library (DLL)   to UNLOAD from the program.

See Also:

DLL GET, DLL LOAD, DLL READ, DLL WRITE, LIST DLL

DLL WRITE

Sets a Dynamic Link Library (DLL) variable to use in the program.

Syntax:

DLL WRITE “*varname*”;*value*

where:

varname is any variable name within the DLL.

value is any numeric value.

Sample:

```
DLL WRITE “Xsection”;3.559
DLL WRITE “Flowrate”;20.9
```

Description:

The DLL WRITE writes the value of a BASIC variable into a Dynamic Link Library (DLL) variable.

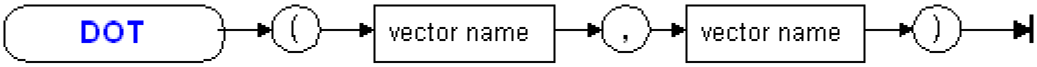
See Also:

DLL GET, DLL LOAD, DLL READ, DLL UNLOAD, LIST DLL

DOT

Returns the dot product of two numeric vectors.

Syntax: DOT(vector, vector)



Sample: `Dotproduct=DOT(Vecx,Vecy)`

`PRINT DOT(X,Y)`

View Sample: DOT.BAS **(also found in examples directory)**

Description:

The dot, scalar or inner product of two vectors is defined to be the product of the magnitudes of the vectors and the angle between them. This is equivalent to the sum of the products of the components of the two vectors.

See Also:

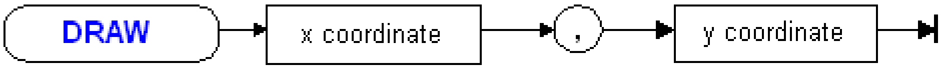
BASE, DET, DIM, MAT, RANK, REDIM, SIZE, SUM

DRAW

Draws a line to the X,Y location.

Syntax: DRAW x-position, y-position

where: x-position, y-position = numeric-expressions



Sample: `DRAW 50,50`

```
DRAW 10,75
DRAW Xx,Yy
```

View Sample: **DRAW.BAS** **(also found in examples directory)**

Description:

A line is drawn from the current position to the specified coordinates using the current line type and pen number. The **DRAW** statement can be used in conjunction with the MOVE statement. **DRAW** always begins with the "pen down" and ends with the pen down. MOVE always lifts the pen before moving to the specified new position. See also PLOT which incorporates pen control into one statement through its syntax. The x-position and y-position arguments express a coordinate in the current SHOW or WINDOW units.

If the arguments of a **DRAW** statement specify a destination point which is outside the clipping rectangle, a theoretical draw to that point is executed. Only that portion of the vector which lies inside the clipping rectangle is drawn. The portion of the vector which lies outside is clipped at the edge of the clipping rectangle.

A **DRAW** to the current position draws a point. The PIVOT statement affects the **DRAW** statement.

See Also:

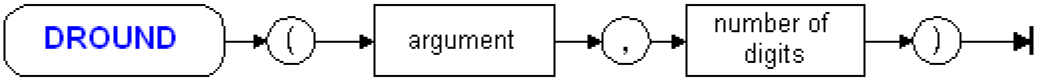
CLIP, IDRAW, IMOVE, IPLOT, LINE TYPE, MOVE, PIVOT, PLOT, RPLOT, SHOW, VIEWPORT, WINDOW

DROUND

Rounds a numeric-expression to the specified number of digits.

Syntax: DROUND(numeric-expression, digits)

where: digits = numeric-expression rounded to an integer.



Sample:

```
Data=DROUND(Sample,10)
PRINT "Current =";DROUND(Amps,4)
```

View Sample: **DROUND.BAS** **(also found in examples directory)**

Description:

If the number of digits is greater than fifteen then numeric-expression is not rounded; if the number of digits is less than one then **DROUND** returns zero.

See Also:

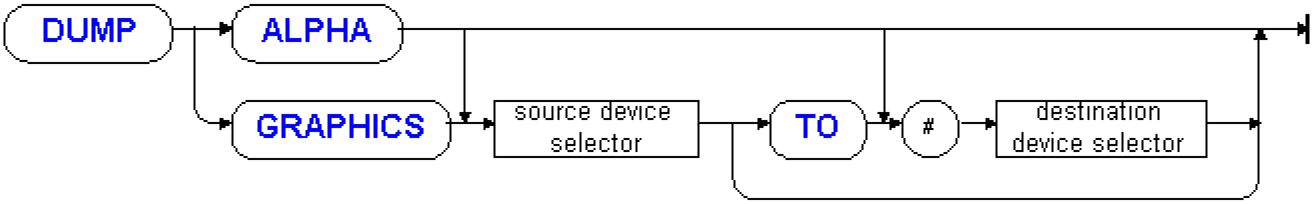
CINT, FIX, FRACT, INT, PROUND, REAL

DUMP

Copies the contents of the display to a printing device.

Syntax: DUMP ALPHA [source [TO #device-selector]]
DUMP GRAPHICS [source [TO #device-selector]]

where: source = device-selector



Sample: `DUMP ALPHA`

```
DUMP ALPHA #702
DUMP GRAPHICS #Dev
DUMP GRAPHICS Color TO #701
```

View Sample: DUMP.BAS **(also found in examples directory)**

Description:

The contents of the **ALPHA** or **GRAPHICS** screen is copied to a printing device. The source, by default, is the CRT. If any other device is specified then no **DUMP** occurs. The **DUMP** is sent to the device specified or to the DUMP DEVICE IS device. Either screen can also be dumped by pressing the DUMP GRAPHICS or DUMP ALPHA keys. To avoid dumping the pseudo-runlight in the lower right-hand corner of the screen, use RUNLIGHT OFF before dumping the screen.

For a **DUMP ALPHA**, alphanumeric characters compatible with any ASCII printer are sent to the printer.

For a **DUMP GRAPHICS**, graphics are sent to the printer in the printer language specified by the CONFIGURE DUMP statement. If no CONFIGURE DUMP is executed, the "WIN-DUMP" driver is used. If MERGE ALPHA WITH GRAPHICS is current, then **ALPHA** text will also be dumped to the printer as part of the graphics data.

Porting Issues

HTBasic supports several types of printers. For this reason, you may need to tell HTBasic what language to use before doing the **DUMP**. The default language is "WIN-DUMP." If you are going to make screen dumps to another type of printer, you must first use the CONFIGURE DUMP statement. You may find it convenient to include this statement in your AUTOST file. Chapter 7, "Printer and Image File Drivers," of the *Installing and Using* manual explains what languages are supported and how to select them.

See Also:

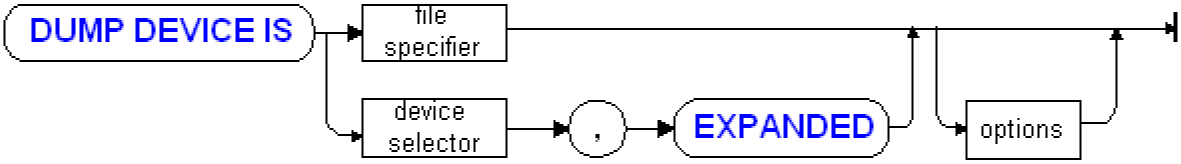
CONFIGURE DUMP, DUMP DEVICE IS

DUMP DEVICE IS

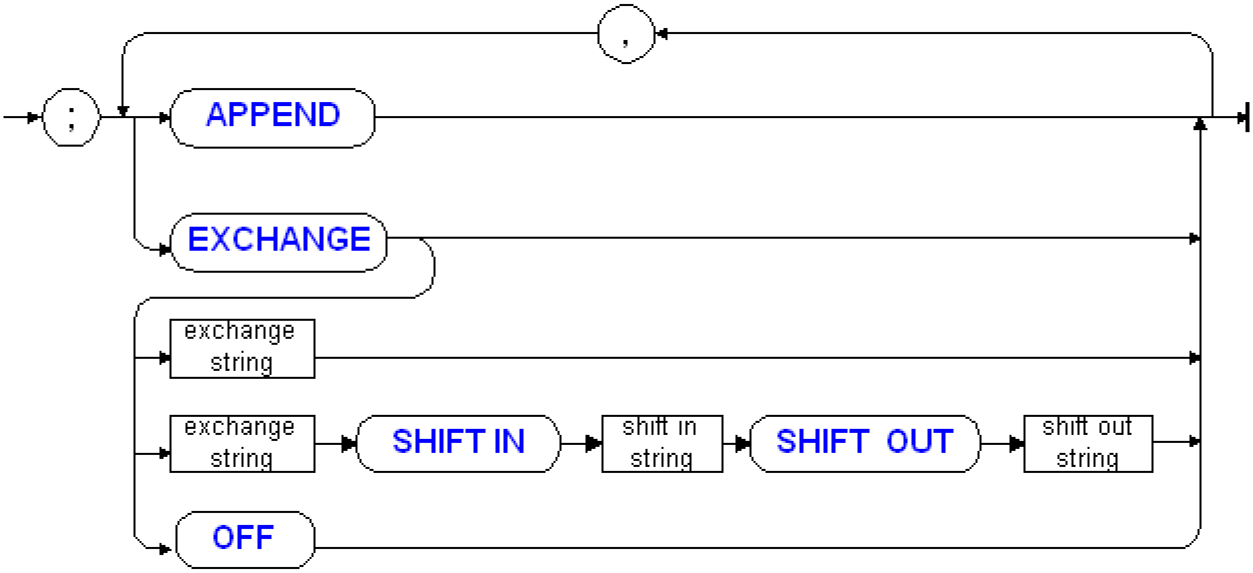
Defines the printing device used by DUMP.

Syntax: DUMP DEVICE IS destination [,EXPANDED] [;APPEND]

where: destination = device-specifier | file-selector



literal form of options:



Sample:

```
DUMP DEVICE IS 10
DUMP DEVICE IS "PICTURE.PCX",EXPANDED
DUMP DEVICE IS "| lpr"
```

View Sample: **DUMP DEVICE IS.BAS** **(also found in examples directory)**

Description:

DUMP DEVICE IS specifies what destination receives the dump data when DUMP ALPHA or DUMP GRAPHICS is executed without a device selector. GINIT resets the destination to the default, which is PRT. Use the CONFIGURE DUMP statement to specify the graphic printer language used.

The number of colors produced in the dump depends on both the display and printer drivers. See CONFIGURE DUMP for more information.

Destinations

The output can be sent to a device (usually a printer), file or pipe. If the destination is a file, it must be an ordinary file or a BDAT file.

Options

If **EXPANDED** is included, the image is rotated by 90 degrees. Depending on the screen and printer types, the image may also be printed larger than when **EXPANDED** is not included.

If **APPEND** is specified and the DUMP is to a file, the file position is moved to the end-of-file before each DUMP. For some DUMP types, multiple images in a file are not supported. For example, the PCX file definition only supports one image per file. If **APPEND** is specified in these cases, the result is undefined. If **APPEND** is not specified, the file is overwritten with each DUMP.

See Also:

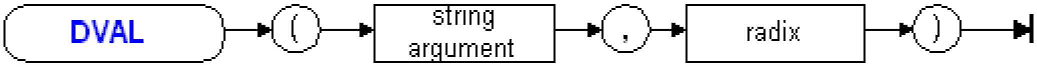
CONFIGURE DUMP, DUMP, PLOTTER IS, RUNLIGHT

DVAL

Converts a binary, octal, decimal or hexadecimal string to a real number.

Syntax: DVAL(string-expression, radix)

where: radix = numeric-expression rounded to an integer



Sample:

```
Value=DVAL(Binary$,Two)
PRINT DVAL("EFA50",16)
```

View Sample: **DVAL.BAS** **(also found in examples directory)**

Description:

DVAL is like VAL, in that a number in string form is converted to numeric form. Unlike VAL, which can only convert decimal numbers, **DVAL** can convert numbers in binary, octal, decimal and hexadecimal.

The string expression contains the number to be converted and the radix must be either 2, 8, 10 or 16. The characters in the string must be legal digits in the specified radix. For example, a binary number can only have characters "0" and "1". Only decimal numbers are allowed to have a minus sign preceding them.

The number expressed in the string is first converted to a 32 bit integer. If the most significant bit is set, the result will be negative. Thus, the string must represent a number within the range of a 32 bit signed integer. The range restrictions are as follows:

Radix	Legal Range
binary	0 through 11111111111111111111111111111111
octal	0 through 37777777777
decimal	-2147483648 through 2147483647
hexadecimal	0 through FFFFFFFF

See Also:

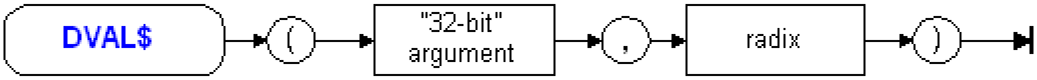
DVAL$, IVAL, IVAL$, VAL, VAL$

DVAL$

Converts a number to a binary, octal, decimal or hexadecimal string.

Syntax: DVAL$(whole-number, radix)

where: whole-number = numeric-expression rounded to a whole number
radix = numeric-expression rounded to an integer



Sample: `Hex$=DVAL$(Number,Sixteen)`

```
PRINT DVAL$(Quantity,8)
```

View Sample: **DVAL$.BAS** **(also found in examples directory)**

Description:

DVAL$** is like VAL$, in that a numeric value is converted to string form. Unlike VAL$, which always expresses numbers in decimal form, **DVAL$ can also express numbers in binary, octal, decimal and hexadecimal form.

Whole-number contains the number to be converted which must be in the range of a 32 bit two's complement integer, -2,147,483,648 through 2,147,483,647. *Radix* must be either 2, 8, 10 or 16.

The converted numbers have leading zeros as necessary to fill unused digit positions. A minus sign is only produced for decimal numbers. The range of numbers produced is the same as those accepted by DVAL.

See Also:

DVAL, IVAL, IVAL$, VAL, VAL$

EDIT

Puts you into program EDIT mode.

Syntax:

```
EDIT [target [,increment]]
EDIT SUB subprogram-name [,increment]
EDIT FN function-name [,increment]
```

where:

target = line-number|line-label|SUB name|FNname
increment = integer constant in the range 1-32766.

Sample:

```
EDIT
EDIT 100,10
EDIT Alabel
EDIT SUB Fire62
EDIT FNPete
EDIT FNOranges
```

Description:

In the syntax above, the space between FN and the function-name is shown for readability. When you type the statement, do not include the space after FN.

The following information applies only to the HTBasic Legacy Editor. For full documentation on the new HTBasic Windows Editor, please see the Installing and Using manual.

The **EDIT** command starts the full screen program editor. It automatically generates and maintains the program line numbers. The default increment for line numbers is 10, but may be specified with the increment value.

If you are editing an existing program, the current edit line will be either the last line edited, the last line with an error or the line specified in the **EDIT** command. You may specify either a line number, line label, SUB program name, or DEF FN function name. If you are editing a new program, the first line number will be 10 unless a line number is specified.

EDIT mode is ended by pressing CLR SCR (HOME on a PC), PAUSE, RUN or STEP keys. It can also be terminated by entering a CAT or LIST command. **EDIT** can only be executed from the keyboard. It cannot be included in a program.

While in **EDIT** mode, the arrow keys, LEFT WORD, RIGHT WORD, PREV, NEXT, BOL, EOL, BEGIN and END keys can be used to move around the program. The INS CHR key toggles the overstrike mode to insert mode and back again. This remains in effect while on the same program line and is reset to overstrike mode when a new line is displayed. The DEL CHR key deletes the character under the cursor. The DEL LEFT key deletes the character to the left of the cursor.

Using the Legacy Editor to insert a line between two program lines or before the first line of the program, position the cursor on the line following the place you wish to insert the new line and then press the INS LN key. If necessary, the program will be partially renumbered and a new line number will be generated for you. You may insert as many program lines as is required. To end the insert line mode press the UP, DOWN, PREV, NEXT, BEGIN, END or INS LN keys. To delete a line, position the cursor on the line you wish to delete and press the DEL LN key.

In the Legacy Editor, the changes to a line are not made permanent until you press ENTER. If you wish to abort the changes, press an arrow key or any other key that moves the cursor to another line.

Keyboard commands can still be entered in **EDIT** mode by first deleting the automatic line number and then entering the command. To delete the line number, backspace over it and then type over the top of it or use the BACKSPACE key to delete back over the top of it or use the END key to clear the current line.

Using keyboard commands you can move a block of text from one place in the program to another (MOVELINES) or copy a block of text from one place to another (COPYLINES). Both of these commands transparently handle any line reference renumbering.

FIND can be used to search for a string of characters. CHANGE can be used to find a string and replace it with another string.

INDENT can be used to automatically indent program constructs. REN can be used to renumber part of or the entire program. DELSUB is used when a subprogram needs to be deleted.

Use "HELP #" to display a list of the keyboard key mappings.

See Also:

CHANGE, COPYLINES, DEL, DELSUB, EDIT KEY, FIND, INDENT, MOVELINES, REN, SECURE, STORE, XREF

EDIT KEY

Puts you into softkey EDIT mode.

Syntax: EDIT KEY key-number

where: key-number = integer constant in the range 0-23.

Sample:

```
EDIT KEY 3
```

Description:

The **EDIT KEY** command edits softkey macros. It is entered by typing **EDIT KEY** n (where n is the softkey number), or by pressing EDIT, the softkey you wish to edit and then the ENTER key. The current definition for the requested key is displayed and the normal editing keys are used to modify the definition (see EDIT). When you are finished press ENTER to save the key definition.

A softkey macro is not available while an ON KEY statement is currently active for that key.

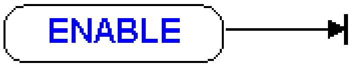
See Also:

EDIT, KBD CMODE, KEY LABELS, KEY LABELS PEN, LIST KEY, LOAD KEY, OFF KEY, ON KEY, READ KEY, SCRATCH, SET KEY, STORE KEY, USER KEYS

ENABLE

Enables all event-initiated branches suspended by DISABLE.

Syntax: ENABLE



Sample: `ENABLE`

Description:

ENABLE does not affect ON END, ON ERROR and ON TIMEOUT.

See Also:

DISABLE, DISABLE INTR, ENABLE INTR, ON, OFF

ENABLE EVENT

Enables HTBasic for Windows to branch upon receipt of a specified event.

Syntax:

ENABLE EVENT @Eventname, Widgethandle

Sample:

```
ENABLE EVENT @Myscrollbar, “CHANGED”
ENABLE EVENT @String1, “KEYSTROKE”
```

Description:

If an ON EVENT statement has been defined for a widget and an event, when the event occurs an event-initiated branch results. Use DISABLE EVENT to temporarily suspend the effects an event would have on your program.

For example, you may want the program to accomplish some task without interruption. After that task is accomplished, you can use **ENABLE EVENT** to re-enable the program’s sensitivity to that event.

While the event is disabled, it can still be logged. Then, when it is re-enabled, the branch will be taken. Only one occurrence of the event will be logged. You must have at least one currently defined event branch in your program to accept inputs from the mouse or keyboard. The events can all be disabled and still accept inputs.

See Also:

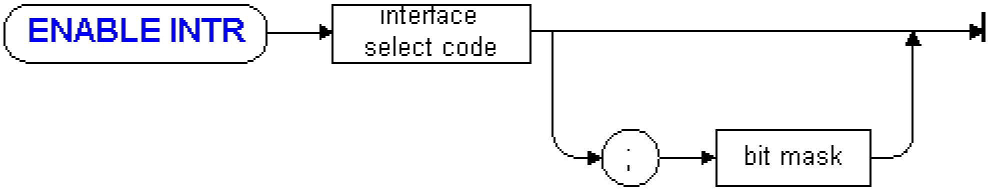
DISABLE, DISABLE EVENT, DISABLE INTR, ENABLE INTR, ON, OFF

ENABLE INTR

Enables interrupts from a specified interface.

Syntax: ENABLE INTR interface-select-code [;enable-mask]

where: enable-mask = numeric-expression rounded to an integer.



Sample:

```
ENABLE INTR 12
ENABLE INTR Isc;Bitmask
```

View Sample: **ENABLE INTR.BAS** **(also found in examples directory)**

Description:

This command enables interrupts from a specified interface for event-initiated branching. An optional bit mask is stored in the interface interrupt-enable register. The default bit mask is the previous bit mask for that interface, or if there is no previous bit mask then a bit mask of all zeros is used. The meaning of the bit mask depends on the interface; consult the interface documentation.

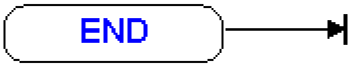
See Also:

DISABLE, DISABLE INTR, ENABLE, ON, OFF

END

Marks the end of the program.

Syntax: END



View Sample: **END.BAS** **(also found in examples directory)**

Description:

An **END** statement is required at the end of the main program. Any subprograms follow the main program **END** statement. Comments may also follow the main program **END** statement.

See Also:

FNEND, SUBEND, PAUSE, STOP

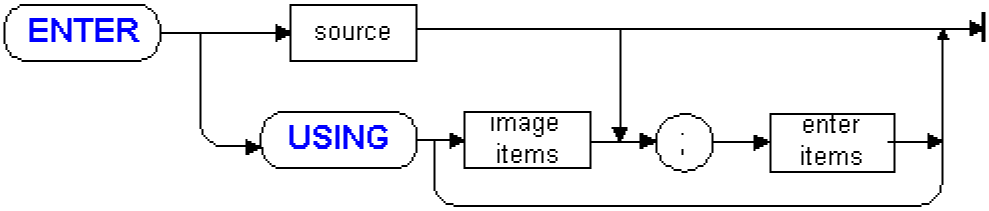
ENTER

Inputs data and assigns it to variables.

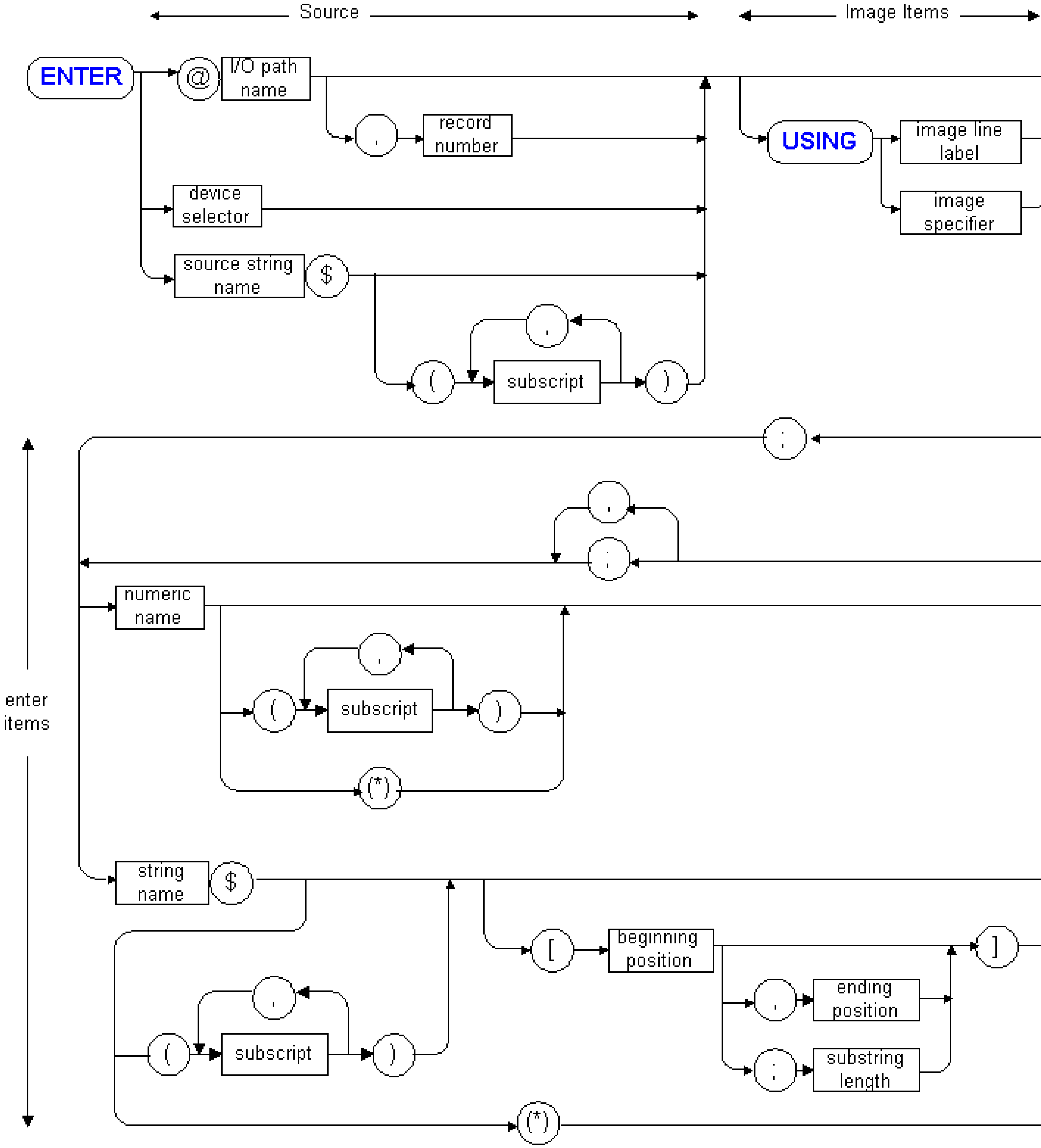
Syntax: ENTER source [USING image] [;item-list]

where: source = @io-path [,record-number] |
device-selector |
string-name$ [(subscripts)]
image = line-number | line-label | string-expression
See IMAGE for the image string syntax.
item-list = item [{,|;} item-list]
item = numeric-name [{(subscripts) | (*)}] |
string-name$ [{[(subscripts)] '['sub-string']' | (*)}]

subscripts = subscript [,subscript...]



expanded diagram:



Sample: ENTER 702;Numeral,Alph$

```
ENTER Dev;P1;P2;P3;P4
ENTER @Picto,Pstr;Array(*)
ENTER @Access USING 20;Lexical$(Def)
```

View Sample: **ENTER.BAS** **(also found in examples directory)**

Description:

Numeric data, array elements or character strings are input from a specified source and the values are assigned to variables. A number builder changes ASCII data to numeric data for assignment to a numeric variable. The number builder ignores blanks and leading non-numeric characters and terminates on the first character received with EOI true or on the first non-numeric character. Arrays may be entered, in row major order, using the full array specifier, "(*)".

String items are terminated with either a line-feed character, a carriage-return/line-feed character pair, an EOI signal or upon filling the dimensioned length of the string. The line-feed or carriage-return/line-feed characters are not entered into the string.

Complex numbers are entered in rectangular form, real part first, followed by imaginary part. The two parts should be separated by EOI or by a non-numeric character.

Sources

File. A file ASSIGNed to an I/O path may be used as the source. An ASCII file is read as ASCII characters. With FORMAT ON, BDAT and ordinary files are also read as ASCII characters. With FORMAT OFF, BDAT and ordinary files are in internal format (see OUTPUT for a description of internal formats). All files may be accessed serially and additionally, BDAT and ordinary files may be accessed randomly by including a record number.

String. A string may be used as the source. **ENTER** begins at the beginning of the string and reads serially. Data is assumed to be in FORMAT ON format.

Device. A device-selector or I/O path may be used as the source to enter items from a device. The default system attributes are used if the source is a device-selector. The ASSIGN statement determines the attributes used if the source is an I/O path. If the device selector is 1, then the source is the CRT. If the device selector is 2, then the source is the keyboard. To terminate a keyboard entry, and append a carriage-return/line-feed, press ENTER. To terminate an entry, with no characters appended, press CONTINUE.

Buffer. A buffer ASSIGNed to an I/O path may be used as the source. The ASSIGN statement determines the attributes used. The buffer empty pointer points to the beginning of the data to be removed and **ENTER** ed. The empty pointer is updated as data is **ENTER**ed.

With USING

See IMAGE for a complete explanation of the image list. The items specified in the image list are acted upon as they are encountered. Each image list item should have a matching enter item. Processing of the image list stops when no matching enter item is found. Conversely, the image list is reused starting at the beginning to provide matches for all remaining enter items. FORMAT ON is used in connection with **ENTER USING**, even if FORMAT OFF has been specified.

Records

When entering from a file, you may specify a record number. The first record in the file is record 1. The record size for **BDAT** files is specified when the file is created and defaults to 256 bytes. For other file types the record size is 1; thus the record number is actually the offset into the file. The first byte of the file is at offset 1. When a record number is specified and the record size is not 1, if the **ENTER** requires more data than a single record, an End of Record error or event occurs.

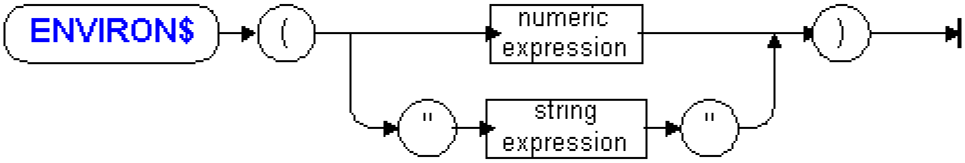
See Also:

IMAGE, INPUT, LINPUT, OUTPUT, PRINT

ENVIRON$

Returns information from the operating system environment.

Syntax: ENVIRON$(string-expression | numeric-expression)



Sample:

```
PRINT "Your path is ";ENVIRON$("PATH")
LOAD ENVIRON$("HTB")&"\autost",1
A$(I)=ENVIRON$(I)
```

View Sample: ENVIRON$.BAS **(also found in examples directory)**

Description:

The **ENVIRON$** function returns the value assigned to an operating system environment variable. You may choose which environment variable to read in one of two ways. If you know the name of a variable, you can specify it by name and its definition will be returned. If the variable does not exist or if the definition is blank, a zero length string is returned. You can also specify a number, in which case both the corresponding variable, an equal sign and the definition are returned. The first variable is number 1. The names of environment variables are case insensitive.

Porting to HP BASIC:

ENVIRON$ is a new HTBasic function that is not available in HP BASIC. It should not be used in programs that must be ported back to HP BASIC.

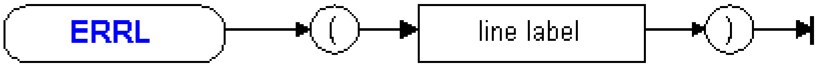
See Also:

COMMAND$, EXECUTE, SYSTEM$

ERRL

Compares a line number with ERRLN.

Syntax: ERRL(line-number | line-label)



Sample:

```
IF ERRL(850) THEN CALL Route_error
IF ERRL(1260) THEN GOTO 5630
IF NOT ERRL(Record) THEN Lock
```

View Sample: **ERRL.BAS** **(also found in examples directory)**

Description:

ERRL returns a 1 if ERRLN is equal to the specified line (in the current context) and 0 otherwise. **ERRL** can be used in IF statements to direct program flow in an error handling routine. **ERRL** is not keyboard executable.

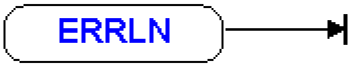
See Also:

CAUSE ERROR, CLEAR ERROR, ERRLN, ERRM$, ERRN, ERROR RETURN, ERROR SUBEXIT, OFF ERROR, ON ERROR

ERRLN

Returns the program line number on which the last error occurred.

Syntax: ERRLN



Sample:
```
PRINT ERRLN
Error1=ERRLN
```

View Sample: **ERRLN.BAS** **(also found in examples directory)**

Description:

The number of the program line on which the most recent error occurred is returned. If no error has occurred, the **ERRLN** function returns 0.

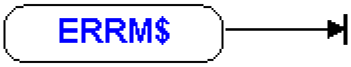
See Also:

CAUSE ERROR, CLEAR ERROR, ERRL, ERRM$, ERRN, ERROR RETURN, ERROR SUBEXIT, OFF ERROR, ON ERROR

ERRM$

Returns the error message text of the last error.

Syntax: ERRM$



Sample:
```
OUTPUT @Errorlog;ERRM$
PRINT ERRM$
```

View Sample: **ERRM$.BAS** **(also found in examples directory)**

Description:

ERRM$ returns the line number (ERRLN), error number (ERRN) and associated error message text. The null string is returned if no error has been generated since start-up, LOAD, GET, SCRATCH or CLEAR ERROR.

Porting Issues

HTBasic error messages are usually similar to those in HP BASIC. Programs that depend on **ERRM$** returning the exact same message as HP BASIC should be modified accordingly. In particular, where an HP BASIC error message has seemed less descriptive than it should be, HTBasic returns a more descriptive message.

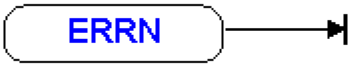
See Also:

CAUSE ERROR, CLEAR ERROR, ERRL, ERRLN, ERRN, ERROR RETURN, ERROR SUBEXIT, OFF ERROR, ON ERROR

ERRN

Returns the last error number.

Syntax: ERRN



Sample:

```
A=ERRN
IF ERRN=75 THEN CALL Exroute
PRINT "Execution Error Number = ";ERRN
10  ON ERROR GOTO 90
20  PRINT X^Y
. . .
80  STOP
90  IF ERRN=27 THEN PRINT "Oops!"
```

View Sample: **ERRN.BAS** **(also found in examples directory)**

Description:

The last program execution error number is returned; or if no error has occurred, a zero is returned. **ERRN**may be used in IF statements to direct program flow in an error handling routine.

Porting Issues

Any error number of 2000 or greater is an HTBasic extension to Rocky Mountain Basic. Not all errors that can occur under HP BASIC can occur under HTBasic. Any error number greater than 10,000 is a Windows error passed to the error handler in HTBasic directly from Windows.

In general, the error numbers returned for errors are the same as those returned by HP BASIC. But in some instances the operating system or environment in which HTBasic runs makes it impossible or impractical to return the same number.

Appendix A contains a list of errors that can occur.

See Also:

CAUSE ERROR, CLEAR ERROR, ERRL, ERRLN, ERRM$, ERROR RETURN, ERROR SUBEXIT, OFF ERROR, ON ERROR

ERROR RETURN

Returns program execution to the line following the most recent error.

Syntax: ERROR RETURN

ERROR RETURN

Sample: `IF Done THEN ERROR RETURN`

View Sample: **ERROR RETURN.BAS** **(also found in examples directory)**

Description:

ERROR RETURN should only be used in connection with ON ERROR GOSUB. A regular RETURN causes the line which generated the error to be re-executed. **ERROR RETURN** skips the line which generated the error and continues execution with the next line.

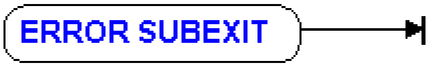
See Also:

CAUSE ERROR, CLEAR ERROR, ERRL, ERRLN, ERRM$, ERRN, ERROR SUBEXIT, OFF ERROR, ON ERROR, RETURN

ERROR SUBEXIT

Returns subprogram execution to the line following the most recent error.

Syntax: ERROR SUBEXIT



Sample:

```
ERROR SUBEXIT
IF Done THEN ERROR SUBEXIT
```

View Sample: **ERROR SUBEXIT.BAS** **(also found in examples directory)**

Description:

ERROR SUBEXIT should only be used in connection with ON ERROR CALL. A regular SUBEXIT causes the line which generated the error to be re-executed. **ERROR SUBEXIT** skips the error line and continues execution with the line following the line in error.

See Also:

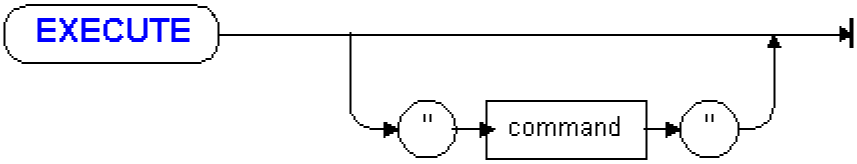
CAUSE ERROR, CLEAR ERROR, ERRL, ERRLN, ERRM$, ERRN, ERROR RETURN, OFF ERROR, ON ERROR, SUBEXIT

EXECUTE

Executes an operating system command.

Syntax: EXECUTE [command] [;option [,option]...]

where: command = string-expression
option = {WAIT OFF | SAVE ALPHA OFF | RETURN numeric-variable}



Sample:

```
EXECUTE "DIR"
EXECUTE "SOL.EXE"
```

View Sample: **EXECUTE.BAS** **(also found in examples directory)**

Description:

The default command interpreter for your operating system is invoked and given the command specified for execution. When the command has completed, control is returned to HTBasic. If the command argument is not specified then the default command interpreter is invoked, you are given a prompt and you may issue one or more commands. You must terminate the command interpreter to return to HTBasic. To return, type "EXIT".

After the command has completed execution, if the **WAIT OFF** option is not specified the message "Hit any key to continue" will be displayed and HTBasic waits until you press any keyboard key. If the **WAIT OFF** option is specified, control immediately returns to the next HTBasic statement.

If the **SAVE ALPHA OFF** option is not specified, the screen is cleared before the command is executed and the screen is restored after the command has finished. If the **SAVE ALPHA OFF** option is specified, the screen is not cleared or restored. Messages written to the screen will write over the current screen. You can, however, redirect the output messages to a file and use the **WAIT OFF** option to prevent writing over the screen.

If the **RETURN** option is specified, the executed program's termination error value is returned in the numeric variable. When control is returned to HTBasic, an attempt is made to service any events which occurred while the command interpreter had control.

When operating under a window system, the **WAIT OFF** and **SAVE ALPHA OFF** options are ignored. To prevent the appearance of a DOS box when running a Windows application use the following syntax:

```
EXECUTE CHR$(13) & "appname.exe"
```

Usage Notes

Windows NT Under Windows NT, CMD is the command interpreter used if no command is specified. To execute a built-in command like "DIR", use "cmd /c DIR". An extension of .EXE is assumed for the command; to execute a .BAT, .CMD or .COM file, include the extension.

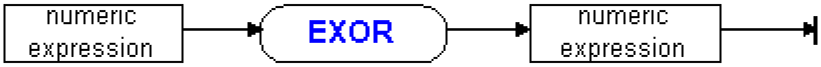
See Also:

QUIT, QUITALL

EXOR

Performs a Logical exclusive OR of two expressions.

Syntax: numeric-expression EXOR numeric-expression



Sample: `I=1 EXOR 0`

```
IF Former EXOR Latter THEN Do
IF A<B EXOR C=D THEN PRINT "ONLY ONE CONDITION IS TRUE"
```

View Sample: **EXOR.BAS** **(also found in examples directory)**

Description:

A **EXOR** B returns a one if exactly one of A or B is non-zero and a zero if A and B are both zero or both non-zero.

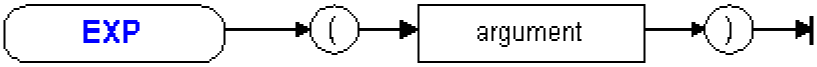
See Also:

AND, OR, NOT

EXP

Returns "e" raised to a power.

Syntax: EXP (numeric-expression)



Sample: `X1=EXP(Y*10)`

```
X2=EXP(-Y^3)
```

View Sample: **EXP.BAS** **(also found in examples directory)**

Description:

EXP returns the value of "e" raised to the power specified by the numeric expression. "e" is the base of the Naperian or Natural logarithm. Its value is approximately 2.718 281 828 459 05.

COMPLEX Arguments

EXP accepts either a COMPLEX or REAL argument and returns a value of the same type. For COMPLEX arguments the real and imaginary parts of **EXP**(Z) are calculated (using real arithmetic) as

REAL(EXP(Z)) = EXP(REAL(Z))*COS(IMAG(Z))
IMAG(EXP(Z)) = EXP(REAL(Z))*SIN(IMAG(Z))

IMAG(Z) specifies radians, regardless of the current trigonometric mode. Notice that intermediate values generated during the calculation of the function can cause over- or underflow errors for very large or small values of Z.

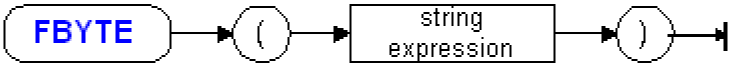
See Also:

LOG, LGT

FBYTE

Checks for first byte of a two byte character.

Syntax: FBYTE(string)



Sample:

```
PRINT FBYTE(A$)
IF FBYTE(A$[I]) THEN PRINT "Two Bytes"
```

Description:

FBYTE is used with SBYTE to determine whether a character is one or two bytes long. FBYTE returns a one if the first byte of the string argument is in the valid range for the first byte of a two byte character.

See Also:

CVT$, SBYTE

FIND

Searches for specified characters in a program.

Syntax: FIND "characters" [IN start [,end]]

where: characters = string-literal
start and end = line-number | line-label

Sample:

```
FIND "PRINT"
FIND "Xx=" IN Math,Result
```

Description:

FIND allows you to search for arbitrary strings in the program. Once found, the program line may be modified or deleted. The search continues after pressing ENTER or DEL LN. If no modification or deletion is needed, pressing CONTINUE searches for the next occurrence. You may exit **FIND** mode by pressing any other function key. The string literal must match exactly. The case of characters is significant.

The **FIND** command from the HTBasic Windows editor input line bings up the **FIND** window and fills fields with old and new values. All other options are ignored.

If start is specified, the search begins with that line. If the line doesn't exist, the line immediately after that line number is used. If a non-existent line label is specified, an error will be reported. If start is not specified, searching will begin with the current line.

If end is specified, the search ends with that line. If the line doesn't exist, the line immediately before that line number is used. If a non-existent line label is specified, an error will be reported. If end is not specified, searching will end with the last line.

FIND is not allowed while a program is running, but it may be used when the program is paused. **FIND** is aborted if a change exceeds the maximum allowable length of a program line or if a line number is altered. **FIND** can only be executed from the keyboard. It cannot be included in a program.

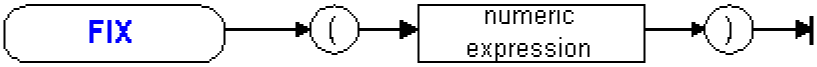
See Also:

CHANGE, COPYLINES, DEL, DELSUB, EDIT, INDENT, MOVELINES, REN, SECURE, XREF

FIX

Truncates a value to INTEGER.

Syntax: FIX (numeric-expression)



Sample: `DRAW FIX(X),Y`

View Sample: **FIX.BAS** **(also found in examples directory)**

Description:

The effect of **FIX** is to remove the fractional part of its argument.

Notice the differences among **FIX**, CINT and INT. **FIX** returns the closest integral value between the REAL value and zero. CINT converts a REAL value to an INTEGER by substituting the closest INTEGER to the value. **FIX** returns the closest integral value between the REAL value and zero. INT returns the closest integral value between the REAL value and negative infinity. Also, CINT actually changes the type from REAL to INTEGER while INT and **FIX** return integral results without changing the type. The following table helps illustrate these differences:

Value x	CINT(x)	FIX(x)	INT(x)
2.6	3	2.0	2.0
2.2	2	2.0	2.0
-2.2	-2	-2.0	-3.0
-2.6	-3	-2.0	-3.0

Porting to HP BASIC:

FIX is a new HTBasic function that is not available in HP BASIC. It should not be used in programs that must be ported back to HP BASIC.

See Also:

CINT, DROUND, FRACT, INT, PROUND, REAL

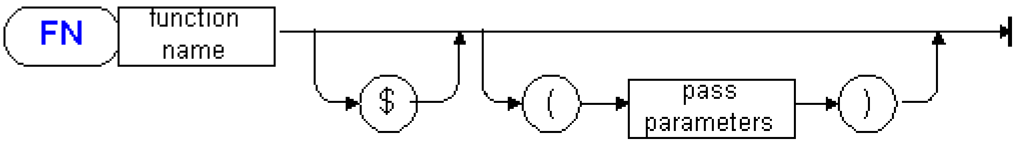
FN

Executes a user-defined function.

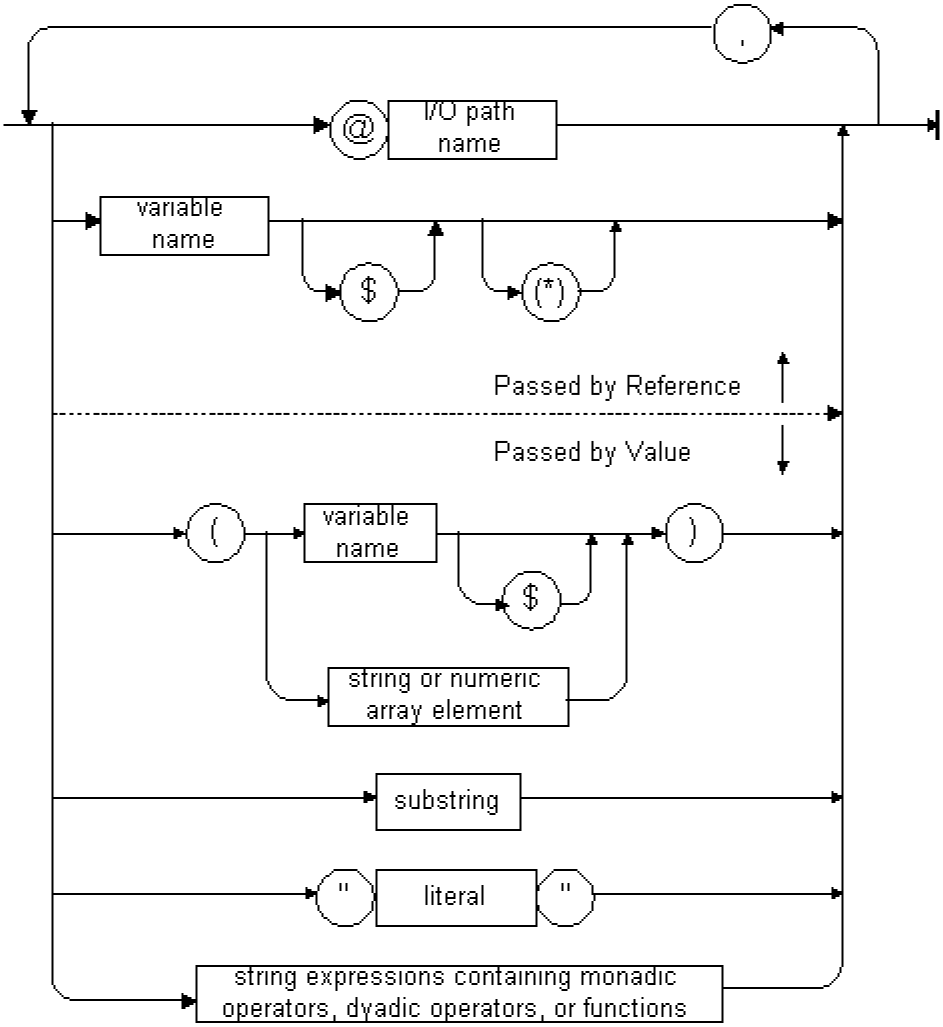
Syntax: FN function-name[$] [(argument [,argument...])]

where: argument = pass-by-reference | pass-by-value
pass-by-reference =
@io-path |
variable-name[$][(*)] |
string-array-element |
numeric-array-element

pass-by-value =
(variable-name[$]) |
(numeric-array-element) |
(string-array-element) |
numeric-constant |
numeric-expression
"string-literal" |
string-name$ [(subscripts)] sub-string |
string-expression



pass parameters:


,
@
I/O path name
variable name
$
(*)
Passed by Reference
Passed by Value
(
variable name
$
)
string or numeric array element
substring
"
literal
"
string expressions containing monadic operators, dyadic operators, or functions

Sample:

```
PRINT "New Value is";FNRate(Y)
Result$=FNCheck$(List$)
Pass=FNDecode(Code,(Express),@Line)
Rotate=FNTranslate(Comp(Trans1+Trans2),Table(*))
```

View Sample: **FN.BAS** **(also found in examples directory)**

Description:

A function subprogram is defined by DEF FN and called by referencing **FN**name. The supplied arguments, if any, may be used in the function's calculations. Upon completion it returns either a string or a numeric value depending on the type of the function name.

Calling a function subprogram changes the program context. Function subprograms may be called recursively. If there is more than one function with the same name the function with the lowest line number is called.

If an expression is defined and evaluated several times throughout a program, it is convenient to define it as a function and then specify the function name instead of the expression. A function can be used anywhere expressions are allowed.

Function subprograms can be included in expressions involved in keyboard calculations. For example, the return value of a function can be displayed by typing the function name and then pressing ENTER.

The arguments specified in the function reference must be of the same type as the parameters in the defining DEF FN. Variables passed by reference must exactly match the DEF FN parameters. Numeric values passed by value are changed to the type (REAL or INTEGER) of the parameter.

See Also:

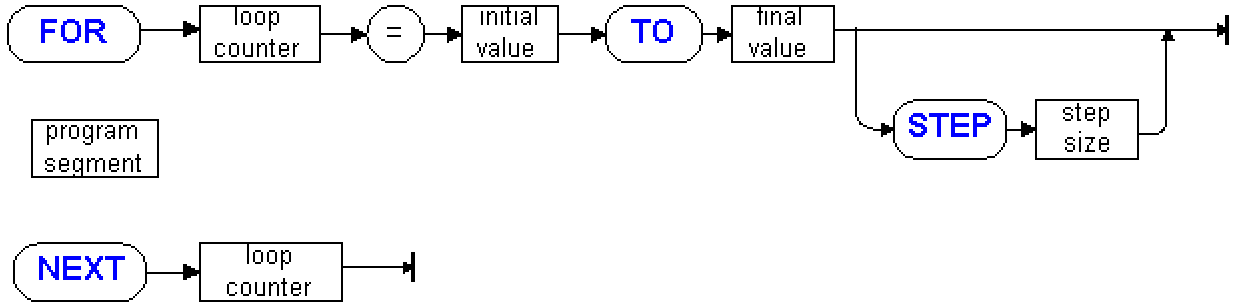
CALL, DEF FN, SUB

FOR ... NEXT

Executes a loop a fixed number of times.

Syntax: FOR control-var = start TO end [STEP step]
statements
NEXT control-var

where: control-var = numeric-name
start, end and step = numeric-expressions
statements = zero, one or more program statements



Sample:

```
10  FOR I=1 TO 100
20    FOR X=1 TO 100
30      PRINT I,X
40    NEXT X
50    FOR J=2*PI TO 0 STEP -PI/100
. . .
80    NEXT J
90  NEXT I
```

View Sample: **FOR NEXT.BAS** **(also found in examples directory)**

Description:

The **FOR ... NEXT** loop is executed a fixed number of times, by incrementing a control variable through a fixed range. The loop consists of statements between the **FOR** and corresponding **NEXT** statement.

When the **FOR** statement is executed, the initial value is assigned to the control variable. The value is then tested against the final value. If it exceeds it (in the proper **STEP** direction) then the **FOR** loop is not executed and control transfers to the line following the matching **NEXT** statement. If there is no **STEP** modifier, the default step size is set to one. The step modifier can be positive or negative. If the step modifier is zero, then the loop is infinitely repeated and no error is generated.

When the **NEXT** statement is executed, the step value is added to the control variable. If the new control value variable is larger than the end value and the step value is positive (or if the new control variable value is smaller than the end value and the step value is negative), the loop terminates and execution continues with the statement following the **NEXT**. If the control variable has not exceeded the end value, then control is returned to the program statement following the corresponding **FOR** statement.

Jumping from outside the **FOR** loop into the **FOR** loop does not give an error but should not be done since the control variable, end value and step value will not be properly set. Jumping from inside the **FOR** loop to outside the **FOR** loop is permitted.

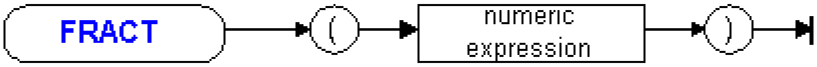
See Also:

CALL, END, FN, GOSUB, GOTO, IF, LOOP, ON, PAUSE, REPEAT, RETURN, RUN, SELECT, STOP, SUBEND, SUBEXIT, WAIT, WHILE

FRACT

Returns the fractional part of an argument.

Syntax: FRACT (numeric-expression)



Sample: `PRINT FRACT(5/3)`

```
Fraction = FRACT(Integer+Fraction)
```

Description:

The **FRACT** function returns a number greater than or equal to zero and less than one. For any value of X, the formula X=INT(X)+**FRACT**(X) is true.

Porting to HP BASIC:

HTBasic allows the **FRACT** of a complex value, returning the fractional part of the real part of the complex value. HP BASIC gives error 620.

See Also:

INT

FRAME

Draws a frame around the clipping area.

Syntax: FRAME

Sample: `FRAME`

Description:

This command frames the clipping area using the current pen and line type. **FRAME** ends with the pen up and positioned in the lower left corner of the frame.

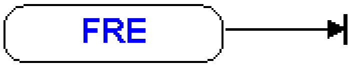
See Also:

AXES, CLIP, GRID, LINE TYPE, PEN, VIEWPORT

FRE

Returns the amount of free memory.

Syntax: FRE



Sample:

```
Remaining=FRE-Needed
IF FRE<Wanted then CALL Wolf
```

View Sample: **FRE.BAS** **(also found in examples directory)**

Description:

This function returns the amount of available memory. To quickly see how much memory is available, type **FRE** and press ENTER. The value will be printed on the message line. This is the same value printed at the end of a LIST statement or returned by the SYSTEM$("AVAILABLE MEMORY") function.

Command Line Switch

The amount of available memory to give HTBasic when it starts is set with a command line switch. The **-w** (workspace) switch specifies how much memory to set aside for your programs and data. The syntax is

`-w` *amount*`[k|m]`

where *amount* should be replaced with a number specifying the amount of memory. *Amount* can optionally be followed by a "k" or an "m". If no "k" or "m" is given, the number specifies bytes. If "k" is given, the number specifies kilobytes and if "m" is given, the number specifies megabytes.

The default workspace size is sixteen megabytes. Note that the amount of free memory reported can be somewhat less than that requested because device drivers or other memory users may allocate some of the memory during startup.

The following example allocates thirty-two megabytes:

```
-w 32M
```

Porting to HP BASIC:

FRE is a new HTBasic function that is not available in HP BASIC. It should not be used in programs that must be ported back to HP BASIC.

See Also:

LIST, SYSTEM$

GCLEAR

Clears the graphics screen.

Syntax: GCLEAR

Sample: `GCLEAR`

View Sample: **GCLEAR.BAS** **(also found in examples directory)**

Description:

If the graphics device is a plotter, **GCLEAR** advances the paper. If the graphics device is a CRT, all planes enabled with the current graphics write-mask are cleared. If any alpha data is present in the same planes, the alpha data is re-written.

For GCLEAR to act as a page eject on some plotters, it is necessary to use HPGL2 mode.

See Also:

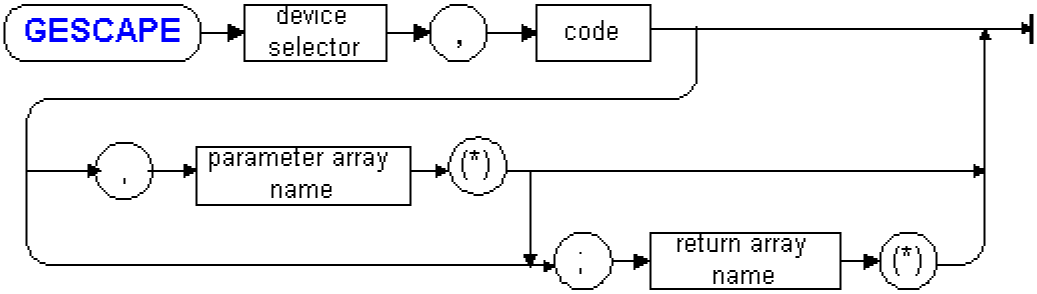
CLEAR SCREEN, GRAPHICS, MERGE ALPHA WITH GRAPHICS, SEPARATE ALPHA FROM GRAPHICS

GESCAPE

Sends device-specific information to a graphic device.

Syntax: GESCAPE device-selector, code [,param(*)][;return(*)]

where: code = numeric-expression, rounded to an integer.
param and return = numeric-array.



Sample: `GESCAPE Dev,Operation;Array(*)`

```
GESCAPE 14,4
GESCAPE 2,3;Hardclip(*)
GESCAPE Plttr,Select,Send(*);Receive(*)
```

View Sample: GESCAPE.BAS **(also found in examples directory)**

Description:

GESCAPE exchanges device-specific data with a graphic device. The code parameter determines what operation will be done. The *param* array sends information to the device. The *return* array receives information from the device. The type, size and shape of the arrays must be appropriate for the requested operation. Codes greater than 99 are extensions to HTBasic which are not present in HP BASIC. Codes in the range 30 to 41 apply to the Windows version only.

Code 1

Return the number of color map entries. The *return* array must be a one dimensional INTEGER array and have at least one element. The first element is assigned the number of color map entries.

Code 2

Return the color map values. The *return* array must be a two dimensional REAL array, must have at least one row, and must have three columns. The first row contains color information for pen 0, second row for pen 1, etc. If the array does not have enough rows or has too many rows, no error is reported. The first column contains the information for red, the second for green and the third for blue. The color information ranges in value from zero to one. Color values are multiples of 1/N, where N is the number of non-black shades available for each color.

Code 3

Return the hard-clip values. The values are returned in plotter units or pixels. The *return* array must be a one dimensional INTEGER array and must contain at least four elements. The first four elements of the array are assigned the values, X min, Y min, X max, Y max, respectively. For a CRT, the fifth and sixth elements give the INTEGER array dimensions needed by the GSTORE command to store the screen image. For example:

```
10  INTEGER A(1:6)
20  GESCAPE CRT,3;A(*)
30  ALLOCATE INTEGER B(1:A(5),1:A(6))
40  GSTORE B(*)
```

Code 4

Set normal drawing mode. Drawing in normal drawing mode with a positive pen number sets each pixel to the pen number. Drawing in normal mode with a negative pen number takes the value of each pixel and clears the bits associated with the pen value. On monochrome displays, the drawing mode is always normal so **GESCAPE** 4 and 5 are not supported.

Code 5

Set alternate drawing mode. Drawing in alternate mode with positive pen numbers performs an inclusive OR on the pen value and the color-map entry number at each pixel. Drawing in alternate mode with negative pen numbers, performs an exclusive OR on the pen value and the color-map entry number at each pixel. On monochrome displays, the drawing mode is always normal so **GESCAPE** 4 and 5 are not supported.

Code 6

Return the graphic display masks. The *return* array must be a one dimensional INTEGER array and must have at least one element. The first element is assigned the value of the graphics write-enable mask. The second element, if present, is assigned the value of the graphics display-enable mask. Each bit in the mask corresponds to one of the bit planes. Bit 0 corresponds to the first plane.

Code 7

Set the graphic display masks. The *param* array must be a one dimensional INTEGER array and must have at least one element. The first element is assigned to the graphics write-enable mask. The second element, if present, is assigned to the graphics display-enable mask. This code is not supported by HTBasic. Often, where operation code 7 is used, MERGE or SEPARATE ALPHA can be used instead.

Window Manipulation

Several GESCAPE codes allow manipulation of the HTBasic windows.

Code	**Operation**
30	Maximize the window
31	Hide the window
32	Restore the window
33	Set interior client of the app window position and size
34	Get interior client of the app window position and size
35	Bring the window to the top
36	Get the screen size
37	Returns the Title Bar enable flag
38	Hide / restore title bar
39	Set the DUMP size (% of paper width)
41	Minimize the window

The following GESCAPE CRT codes have been added for manipulation of the program window.

Code	**Operation**
46	Turn the Toolbar Off
47	Turn the Toolbar On
48	Turn the Status Bar Off
49	Turn the Status Bar On
50	Remove Main Menu
52	Disable Borders on Parent Window
53	Enable Borders on Parent Window
54	Disable Minimize button on Parent Window
55	Enable Minimize button on Parent Window
56	Disable Maximize button on the Parent Window
57	Enable Maximize button on the Parent Window
58	Disable Close button on the Parent Window
59	Enable Close button on the Parent Window
60	Turn the Bookmark Toolbar Off
61	Turn the Bookmark Toolbar On
62	Turn the Debug Toolbar Off
63	Turn the Debug Toolbar On

The following GESCAPE CRT codes have been added for manipulation of the program child window.

Code	**Operation**
130	Maximize the window
131	Hide the window
132	Restore the window
135	Bring the window to the top
137	Returns the Title bar enable flag
138	Hide / Restore the Title bar (Toggle switch)
141	Minimize the window
152	Disable Borders on Child Window
153	Enable Borders on Child Window

The following example shows the syntax for several of the GESCAPES. Note that codes that set information have a comma before the array name while codes that get information have a semicolon.

```
10       INTEGER Get4(1:4),Set4(1:4),Get2(1:2),Set1(1:1)
20       DATA 90,100,500,300      ! Position of upper left corner:
30                              ! 90,100), Width = 500, Height = 300
40       READ Set4(*)
50       GESCAPE CRT,30           ! Maximize the window
60       GESCAPE CRT,31           ! Hide the window
70       GESCAPE CRT,32           ! Restore the window
80       GESCAPE CRT,33,Set4(*)    ! Set position and size: X,Y,W,H
90       GESCAPE CRT,34;Get4(*)    ! Get position and size: X,Y,W,H
100      GESCAPE CRT,35           ! Bring the window to the top
110      GESCAPE CRT,36;Get2(*)   ! Get the screen size: W,H
120      GESCAPE CRT,37;Get3(*)   ! Get the title bar enable flag
130      PRINT Get(2)             ! Print the Screen Size
140      PRINT Get(3)             ! Print the title bar enable flag
150      Set1(1)=50                      ! Set the DUMP size to 50%
160      GESCAPE CRT,38           ! Hide window Title Bar
170      GESCAPE CRT,38           ! Restore window Title Bar
180      Set (1)=50                      ! Set the DUMP size to 50%
190      GESCAPE CRT,39,Set1(*)   ! Set the DUMP size (default is 100%)
200      GESCAPE CRT,41           ! Minimize the window
210      GESCAPE CRT,32           ! Restore the window
220      END
```

Code 103

Returns the current PEN and AREA PEN assignments. The *return* array should be a one dimensional INTEGER array with two elements. The first element is assigned the current PEN assignment. The second element is assigned the current AREA PEN assignment. The following program demonstrates this capability:

```
10  INTEGER P(1)
20  GESCAPE CRT,103;P(*)
30  PRINT "The current      PEN is";P(0)
40  PRINT "The current AREA PEN is";P(1)
50  END
```

Code 104

Sets device-specific information in the PLOTTER IS device. The *param* array must be a one dimensional INTEGER array. The number of elements required depends on the device driver. The first element is the operation number and the subsequent elements are the values associated with that operation.

For the HPGL plotter driver, code 104, operation 1 is used to enable HPGL/2 capabilities. When HPGL/2 is used, polygons are sent to the plotter for rendering. With many plotting devices, this allows the polygons to be filled. When generating an HPGL file for import into other programs, it is often more desirable for the polygon to import as a single unit, rather than a series of lines. To enable HPGL/2, use the following code. Substitute the ISC for the HPGL plotter in place of Isc in line 40.

```
10  INTEGER Param(1)
20  Param(0)=1    ! HPGL Operation Number: 1 = HPGL/2 Flag
30  Param(1)=1    ! Value: 1=enable, 0=disable
40  GESCAPE Isc,104,Param(*)
```

Code 105

Sets device-specific information in the GRAPHICS INPUT IS device. The *param* array must be a one dimensional INTEGER array. The number of elements required depends on the device driver. The first element is the operation number and the subsequent elements are the values associated with that operation.

Code 106

Sets device-specific information in the DUMP DEVICE IS device. The *param* array must be a one dimensional INTEGER array. The number of elements required depends on the device driver. The first element is the operation number and the subsequent elements are the values associated with that operation.

For the dump drivers, code 106, operation 1 is used to specify a portion of the screen to dump when DUMP GRAPHICS is executed. The syntax is:

GESCAPE PRT,106,*param*(*)

The *param* array must be a one dimensional INTEGER array of five elements. The first element is the operation number. The remaining elements specify the boundary for the DUMP. The boundary is specified in screen units:

param(1) - 1
param(2) - Beginning row
param(3) - Ending row
param(4) - Must be 0
param(5) - Must be 0

The CONFIGURE DUMP, PLOTTER IS CRT,"INTERNAL", and GRAPHICS INPUT ISKBD,"KBD" statements reset the row parameters back to the defaults, full screen. The CONFIGURE DUMP statement must be executed before the **GESCAPE** statement. The following program demonstrates this capability:

```
10   INTEGER A(1:5)
20   DUMP DEVICE IS PRT
30   CONFIGURE DUMP TO "HP-PCL"
40   A(1)=1   ! operation code, always 1
50   A(2)=100 ! begin row, screen units
60   A(3)=300 ! end row, screen units
70   A(4)=0   ! reserved, must be 0
80   A(5)=0   ! reserved, must be 0
90   GESCAPE PRT,106,A(*)
100  FRAME
110  MOVE 0,0
120  DRAW 100,100
130  DUMP GRAPHICS
140  END
```

See Also:

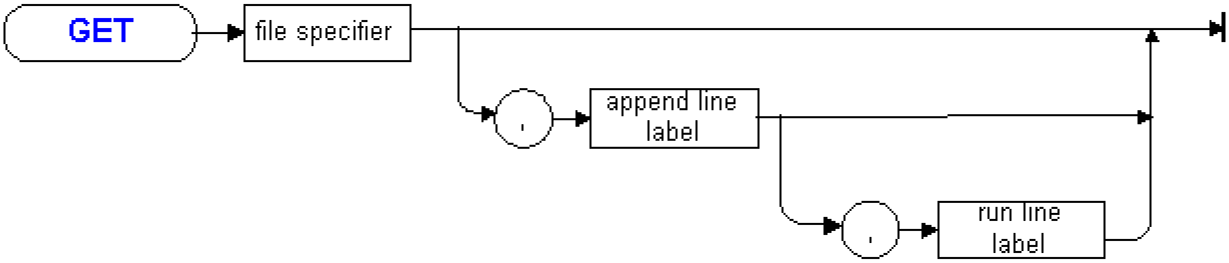
COLOR, GSEND, PLOTTER IS

GET

Loads LIF, DOS or UNIX ASCII program file into memory.

Syntax: GET file-specifier [,append [,run]]

where: append and run = line-number|line-label



Sample: `GET Modprog$,250,20`

```
GET "A:CODEFILE"
GET "Sdir/Cdir/Pdir/CorFile"
GET "GMAT.BAS"
```

View Sample: GET.BAS **(also found in examples directory)**

Description:

When a **GET** is attempted, the first program line is read from the file and checked for a line number. If no line number exists, an error is reported. If **GET** is executed from a running program, this error can be trapped just like any other error. If the first line of the ASCII file has a valid line number, then the **GET** operation first deletes the current program and variables (except for COM variables) and then attempts to read the ASCII program lines into memory. Each line is syntax checked as normal. If a syntax error is found, the line is listed to the PRINTER IS device, turned into a comment (by adding "!* " after the line number) and then saved in memory with the other program lines.

If **GET** specified an append line, then the current program is deleted starting at the append-line; the new lines are appended to the current program and are renumbered to start at the append line number. If **GET** did not specify an append line, then the program is read in without renumbering.

If **GET** specifies a run line (line must be in main context), execution resumes automatically at the run line after a prerun. If **GET**, executed from a program, does not specify a run line, execution resumes at the beginning of the program. If **GET**, executed from the keyboard, does not specify a run line, a RUN command must be given to start execution. If a syntax error occurred during the **GET**, the error is reported and no RUN takes place. These errors cannot be trapped.

GET has been extended to read programs in many different formats: LIF ASCII, DOS ASCII, UNIX ASCII, Viper-I ASCII and Viper-II ASCII. In DOS and UNIX ASCII files, carriage-returns (CR) are ignored and line-feeds (LF) are used to terminate lines. Program lines can be terminated with LF, CR/LF or LF/CR. Files that are terminated with CR only can only be read after an LF is added at the end of each line.

See Also:

CONFIGURE SAVE, LOAD, RE-SAVE, SAVE

GFONT IS

Specifies the font which the LABEL command will use on the graphics screen.

Syntax:

GFONT IS *Fontname*

where:

Fontname is the name of a windows font.

Sample:

```
GFONT IS “Courier”
GFONT IS “Lucida Console”
GFONT IS Fontname$
```

View Sample: **GFONT IS.BAS** **(also found in examples directory)**

Description:

The GFONT IS command allows the user to specify the font that the LABEL command will use on the graphics screen. GFONT IS “” will reset the font to the default font.

Size and rotation of the font are set using the normal label-related commands.

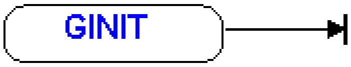
See Also:

LABEL, SYSTEM$

GINIT

Initializes graphics parameters to their default values.

Syntax: GINIT



Sample: `GINIT`

View Sample: **GINIT.BAS (also found in examples directory)**

Description:

GINIT is a fast way to reset colors and other graphic options without explicitly setting each option. **GINIT** also terminates any graphics input device or active plotter. **GINIT**changes the PLOTTER IS back to "INTERNAL." If the previous PLOTTER IS was a file, it is closed. **GINIT** changes the GRAPHICS INPUT IS back to "KBD." The default values for graphic options are dependent on the current device driver, but are typically:

AREA PEN 1
CLIP OFF
CSIZE 5,0.6
LDIR 0
LINE TYPE 1,5
LORG 1
MOVE 0,0
PDIR 0
PEN 1
PIVOT 0
GESCAPE CRT,4

The WINDOW and VIEWPORT are both set to their initial values which are: top = 100, bottom = 0, left = 0, right = RATIO*100. Note the value of the right viewport setting depends on the aspect ratio of the graphic device.

See Also:

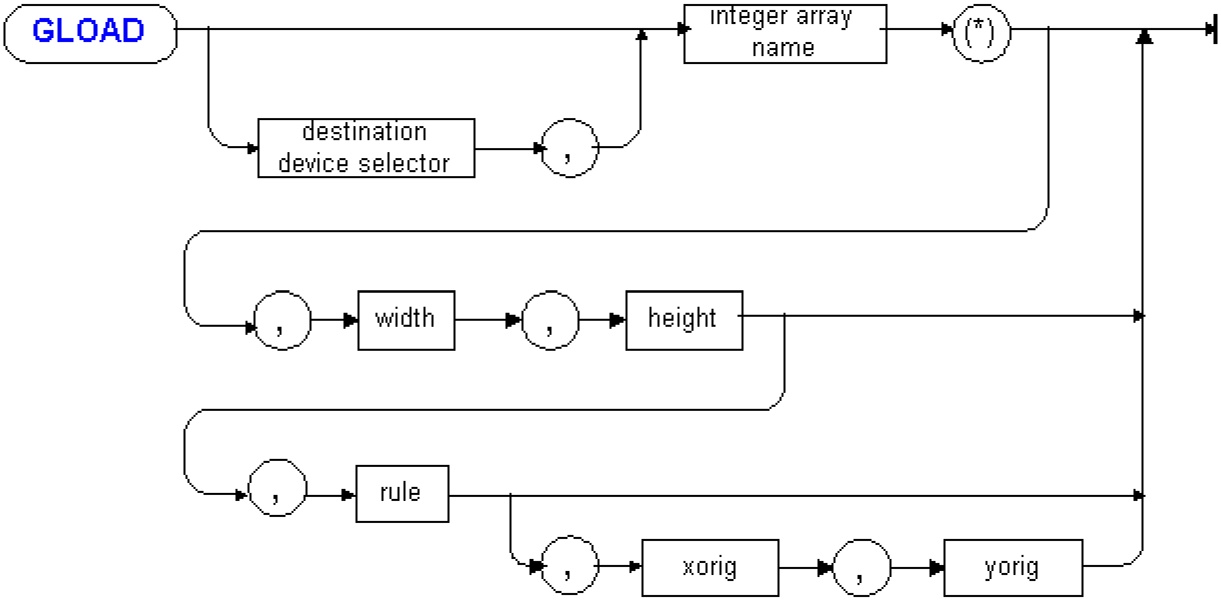
GRAPHICS INPUT IS, PLOTTER IS

GLOAD

Loads an integer array into the CRT display buffer.

Syntax: GLOAD [device-selector,] integer-array(*) [rectangle-params]

where: rectangle-params = ,width,height [,rule [,xorig, yorig]]



Sample:
```
GLOAD Image1(*)
IF Abort THEN GLOAD Explode(*)
GLOAD CRT,Image(*),200,200,3,0,100
```

View Sample: **GLOAD.BAS** **(also found in examples directory)**

Description:

This command displays on the screen an image from an integer array. The image in the array is most frequently one saved from the screen into the array with the GSTORE command. The device-selector specifies the destination device, which must be a bit-mapped device. The CRT is assumed if no device selector is specified.

Two forms of the **GLOAD** statement are supported. The first form is compatible with the **GLOAD** statement in HP BASIC and displays an image which fills the entire screen.

The second form displays an image which fills an arbitrary sized rectangular portion of the screen. For users porting programs from HP BASIC which use the Bstore()/Bload() CSUBs supplied with HP BASIC, the "Porting HP BASIC Programs to the PC" chapter of the *User's Guide*, presents Bstore()/Bload() SUBs which call GSTORE and **GLOAD** using the integrated syntax.

Full Screen GLOAD

The size of the array necessary to store a complete screen image for each display depends on the resolution and on the number of colors the display supports. GESCAPECRT,3 can be used in a program to determine the size necessary. The following table gives the sizes for some display adaptors. The array may be declared larger or smaller than the size given. If the array is not large enough to contain a full screen image, **GLOAD** stops when all the array contents have been transferred to the screen. If the array is too large, only part of the array will be used. If an attempt is made to **GLOAD** an image to a display that is different from the GSTORE display, unpredictable results will occur. If the color map has different values than when the image was GSTOREd, the colors will not match the original image.

Display	**Array Size**
SVGA16;640x480	Image(1:160,1:480)
SVGA16;800x600	Image(1:200,1:600)
SVGA16;1024x768	Image(1:256,1:768)
SVGA256;640x480	Image(1:320,1:480)
SVGA256;800x600	Image(1:400,1:600)
SVGA256;1024x768	Image(1:512,1:768)

The format of the image data within the array is documented for most displays in the *User's Guide*.

Rectangular Blocks

When a *Width* and *Height* are specified after the image array, only a rectangular block is loaded from the array onto the display. *Width* and *Height* are specified in pixels. Optionally, a *Rule* can be specified which instructs **GLOAD** how to combine the contents of the array with the contents of the screen. Presently, only a value of 3 is supported, which causes the contents of the array to totally overwrite the specified block on the display. The block will be located with the upper left corner at the current graphic position. Alternately, a position can be specified with the *Xorigin*, *Yorigin* parameters. These parameters should be specified in the current WINDOW units, not pixels or VIEWPORT units (GDUs).

The image is stored with one byte per pixel. This makes images somewhat transportable among different displays. It also means that the number of elements necessary to store the image is equal to Width*Height/2. If the width is even, the array could be declared as

```
INTEGER Image( 1:Width/2,1:Height)
```

If the array is too small, an error is given. If the array is too large, the extra elements are ignored. If **GLOAD**is used to display an image on a display with less colors than the GSTORE display, the results are undefined. If the color map is different than the color map in effect when the image was GSTOREd, the colors will not match the original image.

Windows Version Usage Notes

Not all windows CRT drivers support **GLOAD**/GSTORE. Full screen **GLOAD**/GSTORE uses BMP format. The contents of the array can be saved in a file and modified by most Windows draw/paint programs. The array contains both palette and image information.

graphics_buffer off. If the graphics_buffer command line switch is off and another window overlaps the HTBasic window, the overlapping portion of the window will be included in the stored image. If the window is iconified, the stored image will be the HTBasic icon. If part of the HTBasic window is offscreen, only the part on screen is stored. To avoid these side-effects, use the "-gr on" command line switch.

COLOR LOSS. If a BMP file is loaded into an array and **GLOAD**ed to the screen, some color information may be lost. Any color in the image that doesn't exist in the destination palette are changed to similar colors that do exist in the palette.

See Also:

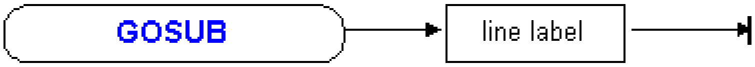
GESCAPE, GSTORE

GOSUB

Transfers control to a subroutine.

Syntax: GOSUB subroutine

where: subroutine = line-label | line-number



Sample: `GOSUB 1000`
`GOSUB John`

View Sample: GOSUB.BAS **(also found in examples directory)**

Description:

A subroutine is any portion of a program context beginning with a line mentioned in and defined in the same context, as a **GOSUB** statement and ending with a RETURN statement.

When a running program encounters a **GOSUB** statement, it saves the current line number and then transfers control to the specified line. Execution continues normally until a RETURN statement is executed, at which point the program jumps back and resumes execution at the line after the **GOSUB** statement. Execution of a RETURN statement without a **GOSUB** will give an error.

If the subroutine is called by ON ERROR GOSUB, it can also include ERROR RETURN statements. A RETURN re-executes the statement which caused the error, while ERROR RETURN skips it.

Porting Issues

Under HTBasic, **GOSUB** and ALLOCATE use the same stack. Intermixing these statements can cause changes in available memory that are different from HP BASIC. In practice this causes no problems.

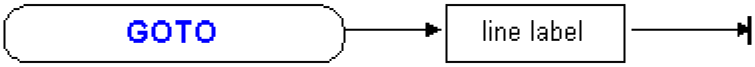
See Also:

ERROR RETURN, GOTO, ON, ON-event GOSUB, RETURN

GOTO

Transfers control to a specified line.

Syntax: GOTO { line-label | line-number }



Sample:

```
GOTO 510
GOTO Loop
```

View Sample: **GOTO.BAS** **(also found in examples directory)**

Description:

Program execution continues at the specified line. This line must be in the current context.

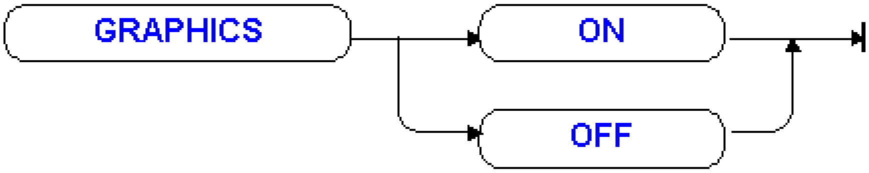
See Also:

GOSUB, ON

GRAPHICS

Makes the graphics screen visible or invisible.

Syntax: GRAPHICS { ON | OFF }



Sample:
```
GRAPHICS ON
IF No_show THEN GRAPHICS OFF
```

View Sample: **GRAPHICS.BAS** **(also found in examples directory)**

Description:

GRAPHICS ON makes the graphics screen visible; **GRAPHICS OFF** makes it invisible.

GRAPHICS ON/OFF has no effect when ALPHA and **GRAPHICS** are MERGEd. SEPARATE ALPHA FROM GRAPHICS must be executed before this statement has any effect.

See Also:

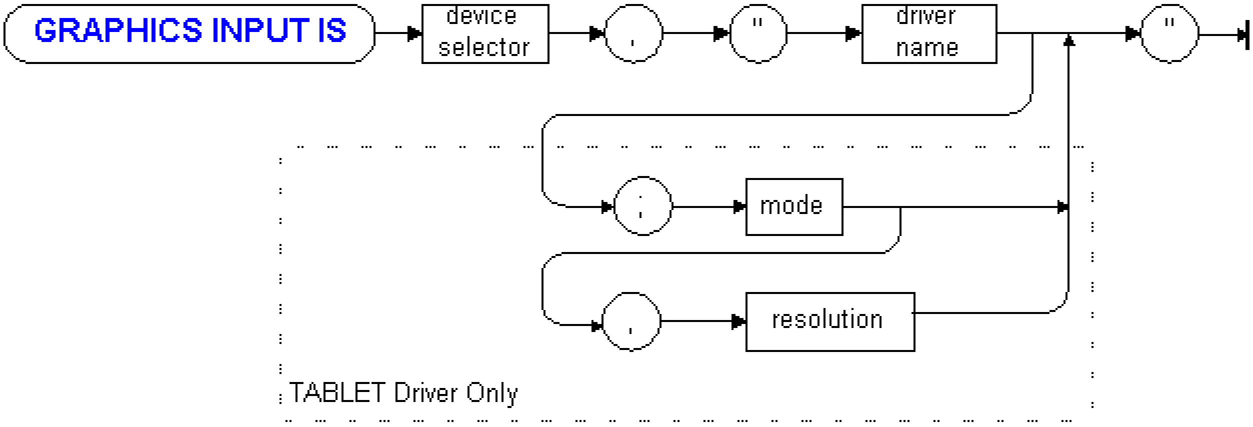
ALPHA, GCLEAR, MERGE ALPHA WITH GRAPHICS, PLOTTER IS, SEPARATE ALPHA FROM GRAPHICS

GRAPHICS INPUT IS

Defines the device to be used for graphic input.

Syntax: GRAPHICS INPUT IS device-selector, "driver-name [;options]"

where: driver-name = KBD | HPGL | TABLET
options = driver options. See text for detailed information.



Sample:

```
GRAPHICS INPUT IS KBD,"KBD"
GRAPHICS INPUT IS 705,"HPGL"
GRAPHICS INPUT IS 705,"TABLET;BIN-2,0,5000,0,5000"
```

View Sample: **GRAPHICS INPUT IS.BAS** **(also found in examples directory)**

Description:

This statement specifies which device and driver to use for DIGITIZE, READ LOCATOR and SET LOCATOR statements.

The device-selector specifies the device or interface to use to communicate with the graphic input device. This is usually KBD, an IEEE-488 device selector or the Serial interface select code. The driver name and options, shown in literal form in the above syntax diagram, can be specified with a string expression. The string specifies which driver to use with the device. The default device is KBD and the default driver is "KBD".

Graphics Input Drivers

HTBasic supports loadable graphics drivers. The first time a driver is specified in a **GRAPHICS INPUT IS** statement, the driver is loaded and used for graphics input. When the driver is subsequently specified, it is not loaded again, but is again used for graphics input. The following table lists the drivers available at the time of this manual printing. (Not all drivers are available in all versions.)

Name	For These Devices
KBD	Keyboard arrow keys or Mouse
HPGL	HPGL Plotters or Digitizers
TABLET	Most available digitizing tablets

HTBasic automatically loads the "KBD" driver when it starts. Up to ten graphic and dump drivers can be loaded at a time.

Driver files can be loaded at any point. It is recommended that **GRAPHICS INPUT IS** statements be included in your AUTOST file to load any necessary drivers.

To find the driver file HTBasic takes the driver specified in the **GRAPHICS INPUT IS** statement and performs several operations upon it to find the correct file. ".DW6" is appended to the name. Then the following locations are searched, in the specified order:

1. The directory containing the HTBasic executable.

2. The current directory.
3. The Windows system directory (such as \WINNT\SYSTEM32).
4. The Windows directory.
5. The directories listed in the PATH environment variable.

KBD Driver

The keyboard (KBD) graphics input driver provides support for input of X and Y coordinates from the keyboard arrow keys or the mouse. The KBD driver is loaded at start up. The command to switch back to the KBD graphics input driver from another driver is

```
GRAPHICS INPUT IS KBD,"KBD"
```

The following example program shows how to set up the KBD driver and get coordinate information from the input device.

```
10  PLOTTER IS CRT,"INTERNAL"
20  GRAPHICS INPUT IS KBD,"KBD"
30  TRACK CRT IS ON
40  FRAME
50  DIGITIZE X,Y,S$
60  PRINT X,Y,S$
70  END
```

HPGL Driver

The HPGL graphics input driver provides support for any input device that accepts Hewlett Packard's HPGL language. Some HPGL compatible devices are the HP 9111A and HPGL plotters.

TABLET Driver

The TABLET graphics input driver provides support for most digitizers currently available. It usually uses either the serial port or the IEEE-488 (GPIB) bus to communicate with the tablet. The following guidelines will help you in loading the driver and in selecting the proper tablet configuration and data communication options. The command to load the TABLET graphics input driver is:

GRAPHICS INPUT IS Isc,"TABLET;[*mode*[,]][*resolution*]"

The *mode* option allows you to specify the method in which the tablet's data is interpreted by the driver. If both mode and resolution options are specified, specify the mode option first and separate the two by a comma. The following table gives the legal values for mode:

Mode	**Meaning**
(None)	Comma separated ASCII
BIN-1	Summagraphics MM Binary Format
BIN-2	Hitachi Binary Format
BIN-3	UIOF Binary Format.

If no mode is specified, then the driver assumes the tablet is using a comma separated, CR/LF terminated, ASCII data format. The data cannot contain any decimal points within the string. ASCII format is preferred over binary; it tends to be easier to setup and get working. The binary formats are explained in greater detail in the *Installing and Using* manual. The *resolution* option is sometimes necessary to scale X and Y values read from the tablet. The TABLET driver assumes a default maximum resolution of 11000 units in both the X and Y directions. This value is used to scale the digitizer coordinates to the display WINDOW coordinates. If this value is not correct for your digitizer or if you want to adjust for any distortion, you can change the scaling values with the following command:

GRAPHICS INPUT IS 9,"TABLET;Xmin,Xmax,Ymin,Ymax"

Xmin and Xmax are the digitizer's X values that correspond to the display's minimum and maximum X values respectively. Ymin and Ymax are the digitizer's Y values that correspond to the display's minimum and maximum Y values. Please note that these values are specified in device units.

The TABLET driver scales the digitizer X and Y coordinates into the display WINDOW coordinates. For example, suppose the screen's WINDOW resolution is 0-133 in the X direction and 0-100 in the Y direction and the digitizer's X and Y resolution is 0-11000. If the digitizer returns 11000,11000 as the current X and Y location, the DIGITIZE statement will return a value of 100,133 to the user. If you want the X and Y values to be the same for equal movements in the X and Y directions, specify a square WINDOW. For example:

```
WINDOW 0,100,0,100
```

The digitizer has two options that are critical to make it work properly with HTBasic. They are as follows:

- Handshaking Mode
- Absolute coordinates

Some other tablet settings that are not critical, but recommended are as follows:

- Data transmitted only in proximity.
- Disable Increment mode.
- Disable leading zero's.
- Enable RUN mode.
- Enable Maximum report rate.

Please consult your digitizer documentation for the correct switch settings for these options.

Communication

The TABLET and HPGL drivers usually use either the serial port or the IEEE-488 (GPIB) bus to communicate with the digitizer. This is specified by the device-selector in the **GRAPHICS INPUT IS** statement. For example:

```
GRAPHICS INPUT IS 702,"TABLET" !GPIB Address 2
GRAPHICS INPUT IS 9,"TABLET"   !First Serial Port
```

Communication with the tablet over the GPIB bus is straight forward. You specify the device-selector (i.e. 702) and the control and data messages proceed without further setup.

Communication with the tablet over the serial port is more involved because of the many serial configuration options. The SERIAL driver defaults to 8 Data Bits, No Parity Bit, 1 Stop Bit and a speed of 9600 Baud. Make sure that the switches on the tablet are set to match these defaults or specify the differences when loading the SERIAL driver.

The tablet may support either XON/XOFF handshaking or hardware handshaking. Find out which method your tablet supports and set the SERIAL driver to use the same handshaking. By default the SERIAL driver uses XON/XOFF handshaking, the following line is all that is needed to set the driver to this method.

```
10  LOAD BIN "SERIAL" !Loads SERIAL device driver
```

If you need to use hardware handshaking, you will have to set a number of other registers within the SERIAL driver. The following program lines specify hardware handshaking.

```
10  LOAD BIN "SERIAL" !Loads SERIAL device driver
20  CONTROL 9,5;0     !Use DTR and RTS
30  CONTROL 9,12;0    !Read DSR, CD and CTS
40  CONTROL 9,100;0   !Disable XON/XOFF handshaking
```

With some digitizers the RTS line must be held active to make the TABLET driver work correctly, otherwise an error will occur after several successful reads. To hold the RTS line active change program line 20 to CONTROL 9,5;2. Make sure the tablet is set to hardware handshaking. For some tablets, this is specified as CTS handshaking.

Porting Issues

Both HP BASIC and HTBasic do an implicit **GRAPHICS INPUT IS** assignment for you if you attempt to use graphic input statements before an explicit **GRAPHICS INPUT IS** statement. The difference is that HTBasic does the implicit **GRAPHICS INPUT IS** as soon as HTBasic is started and HP BASIC waits until the first graphic input statement is executed. The only known effect of the different approach is that under HP BASIC, a SYSTEM$("GRAPHICS INPUT IS") returns "0" until the first graphic statement is executed and HTBasic returns the correct value anytime.

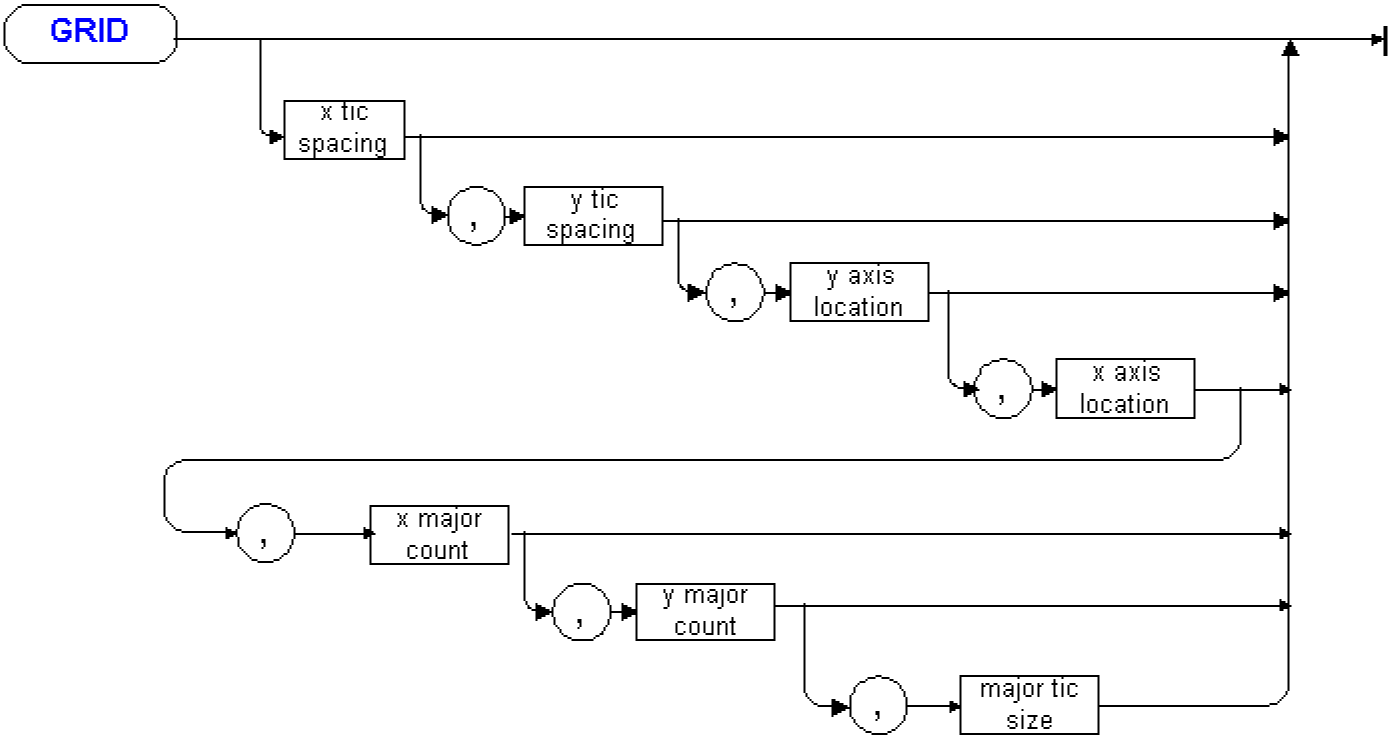
See Also:

DIGITIZE, PLOTTER IS, READ LOCATOR, SET LOCATOR, TRACK

GRID

Draws a grid pattern.

Syntax: GRID [x1 [,y1 [,x2 [,y2 [,x3 [,y3 [,minor]]]]]]]



Sample: `GRID 20,20`
`GRID 20,20,0,0,2,2`

View Sample: **GRID.BAS** **(also found in examples directory)**

Description:

With no arguments **GRID** produces a simple axes. The addition of x1 and y1 cause a grid to be drawn. The x1,y1 values specify the spacing between grid lines.

A value of zero (the default) disables grid lines in that direction. Grid lines are drawn across the entire soft-clip area. The values x2,y2 specify the origin of the grid; the defaults are 0,0.

The values of x3,y3 substitute short tick marks in the place of full grid lines. A value of n specifies that only 1 out of n divisions use a full grid line. The other (n-1) divisions use tick marks instead. The defaults are 1,1. This disables tick marks because full grid lines are drawn for all the divisions.

The minor value specifies the size of tick marks. The default is 2 graphic display units.

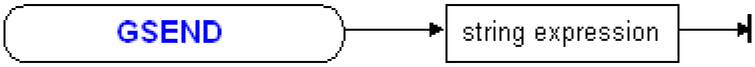
See Also:

AXES, FRAME, LINE TYPE, PEN

GSEND

Sends commands to the PLOTTER IS device.

Syntax: GSEND string-expression



Sample:

```
GSEND Msg$
IF Aplotter THEN GSEND "PD;"
```

Description:

This command sends a string to the current PLOTTER IS device. This is sometimes useful in order to send a command to the PLOTTER IS device which is not normally sent by the graphic statements.

See Also:

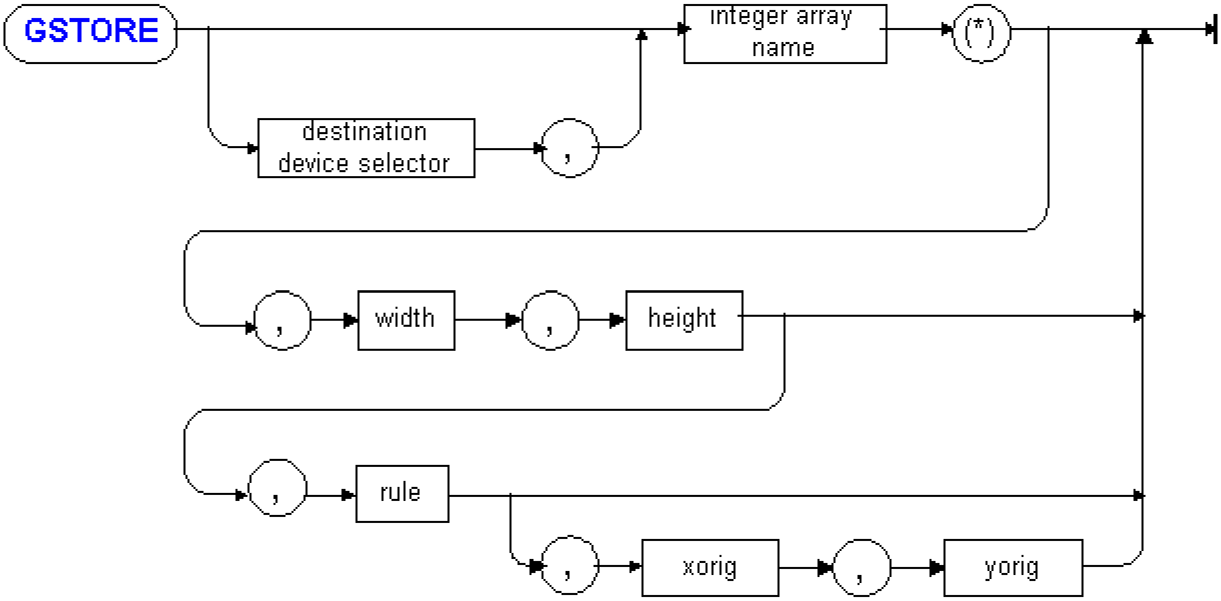
GESCAPE, PLOTTER IS

GSTORE

Stores the CRT display buffer into an integer array.

Syntax: GSTORE [device-selector,] integer-array(*) [rectangle-params]

where: rectangle-params = ,width,height [,rule [,xorig, yorig]]



Sample: `GSTORE Diagram(*)`

```
IF Keep THEN GSTORE Current(*)
GSTORE CRT,Image(*),200,200,3,0,100
```

View Sample: **GSTORE.BAS (also found in examples directory)**

Description:

This command saves an image from the screen into an integer array. The image in the array is most frequently used for re-display with the GLOAD command. The device-selector specifies the source device, which must be a bit-mapped device. The CRT is assumed if no device selector is specified.

Two forms of the **GSTORE** statement are supported. The first form is compatible with the **GSTORE** statement in HP BASIC and stores an image which fills the entire screen.

The second form stores an image which fills an arbitrary sized rectangular portion of the screen. For users porting programs from HP BASIC which use the Bstore()/Bload() CSUBs supplied with HP BASIC, the "Porting HP BASIC Programs to the PC" chapter of the *User's Guide*, presents Bstore()/Bload() SUBs which call **GSTORE** and GLOAD using the integrated syntax.

See Also:

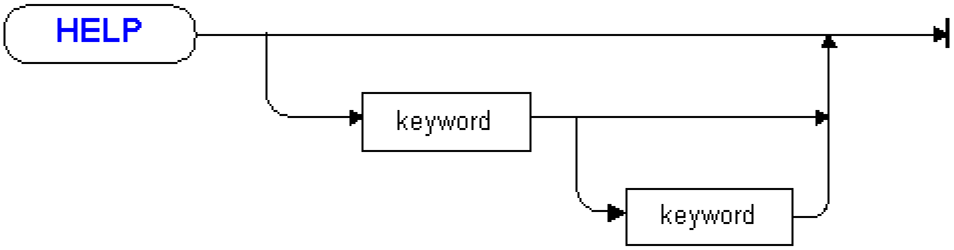
GESCAPE, GLOAD

HELP

Displays Manual pages on the computer screen.

Syntax: HELP [manual-entry [second keyword]]

where: manual-entry = a keyword from the manual
second-keyword = legal secondary keyword



Sample:

```
HELP
HELP SELECT
HELP CONFIGURE LABEL
```

Description:

The **HELP** command is used to look up material in this online help. The online manual is virtually the same as the printed material.

To look up a *manual-entry* when not in **HELP** mode, type:

HELP *manual-entry*

and press ENTER. The first page about that manual entry will be displayed. A primary keyword may have several manual entries, describing different combinations of the keyword followed by a secondary keyword. For example, the primary keyword ON has several entries, such as ON, ON CYCLE, ON DELAY, etc. The

```
HELP ON
```

command places you at the start of the first entry that talks about ON. The

```
HELP ON TIMEOUT
```

command places you at the start of the ON TIMEOUT entry.

Navigating in HELP Mode

To switch to a different manual entry while in HELP mode, type the new keyword and press ENTER. To get another page of information, press ENTER or CONTINUE. To exit the **HELP** mode, press CLR SCR. To read something that has scrolled off the top of the screen, scroll the screen back using PREV and NEXT or the UP and DOWN arrow keys.

Navigating in Windows Help

The Windows version of HTBasic uses the standard Windows Help system used by most windows programs. The buttons and menu items at the top of the help system do the following:

Use this	**To do this**
Contents	View the table of Contents
Search	Search the index
Back	Return to previously viewed topics
History	View list of previously viewed topics
<<	View the previous page of the manual
>>	View the next page of the manual
Print	Print the current topic
Copy	Copy the current topic to the clipboard
Annotate	Attach a note to the current topic
Bookmarks	Place a bookmark, or go to a bookmark
Always on Top	Force Help window to stay on top of other windows

Additionally, hyperlinks allow easy navigation among related topics. On most displays, hyperlinks are underlined in green. Click on a link to show the related topic. Click on Back to return to the previous topic.

HIL

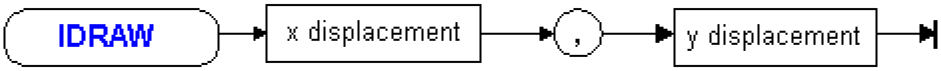
HIL related statements are not supported.

IDRAW

Draws a line an incremental distance.

Syntax: IDRAW x-displacement, y-displacement

where: x-displacement and y-displacement = numeric-expressions



Sample:

```
IDRAW 0,25
IDRAW DispX,DispY
IDRAW X+10,Y+25
```

View Sample: **IDRAW.BAS** **(also found in examples directory)**

Description:

The pen is lowered and then moved to the position calculated from adding the specified X and Y displacement to the current pen position. After **IDRAW** executes, the logical pen position is updated and the pen is left in the down position. **IDRAW** 0,0 draws a point.

If you specify a destination which is outside the clipping area, the logical position is set to that point but the pen is not moved. Only the portion of the vector which lies inside the clipping area is plotted.

The PIVOT statement affects the **IDRAW** statement.

See Also:

CLIP, DRAW, IMOVE, IPLOT, LINE TYPE, MOVE, PIVOT, PLOT, RPLOT, SHOW, VIEWPORT, WINDOW

IF ... THEN

Performs an action if a condition is true.

Syntax:

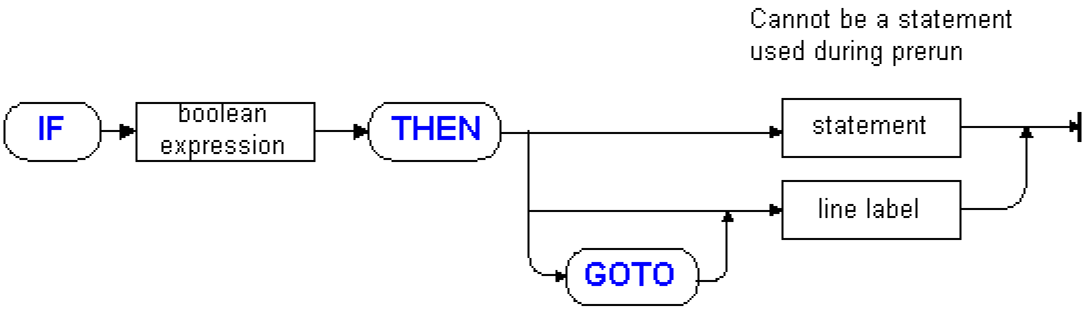
Single Line IF:
IF expression THEN action

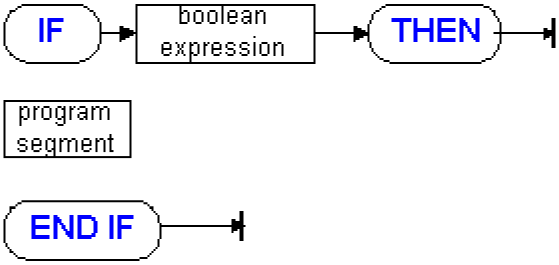
Block IF:

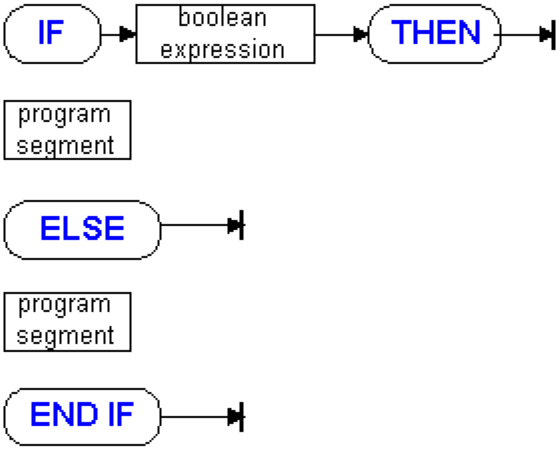
```
IF expression THEN
    statements
[ELSE]
    statements
END IF
```

where:

expression = numeric-expression rounded to a boolean true if non-zero and false if zero.
action = line-number | line-label | program statement
statements = zero, one or more program statements







Sample:

```
10 IF J2=K THEN 1200
20 IF X=Y THEN Y=Z
30 IF A<0 THEN
40   PRINT "Below Limit!"
50 ELSE
60   CALL Convert
70 END IF
```

View Sample: IF THEN.BAS **(also found in examples directory)**

Description:

In a single line **IF** statement, if the expression is true, the action following the **THEN** is taken. If the expression is false, execution continues with the statement following the **IF** statement.

The following statements are not allowed in single line **IF ... THEN** statements:

CASE
DATA
ELSE
END LOOP
EXIT IF
IF
LOOP
REAL
SELECT
UNTIL
CASE ELSE
DEF FN
END
END SELECT
FNEND
IMAGE
NEXT
REM
SUB
WHILE
COM
DIM
END IF
END WHILE
FOR
INTEGER
OPTION BASE
REPEAT
SUBEND

To construct a block **IF** statement, no action is allowed after the **THEN** on the **IF** statement and the block structure must end with an **END IF** statement. Only the block **IF** statement allows the optional **ELSE** statement. If the expression is true the statements between the **IF ... THEN** and the **ELSE** are executed. Control then continues with the statement following the **END IF** statement. If the expression is false, the statements between the **ELSE** and the **END IF** are executed.

Although HTBasic does not have an explicit ELSE IF statement, it is possible to accomplish the same thing using a SELECT statement. See SELECT for an example.

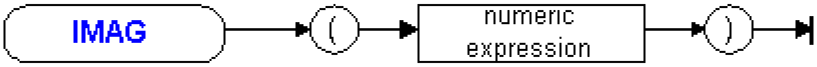
See Also:

CALL, END, FN, FOR, GOTO, GOSUB, LOOP, ON, PAUSE, REPEAT, RETURN, RUN, SELECT, STOP, SUBEND, SUBEXIT, WAIT, WHILE

IMAG

Returns the imaginary part of a complex number.

Syntax: IMAG(numeric-expression)



Sample:

```
PRINT IMAG(Z)
DRAW REAL(C),IMAG(C)
```

View Sample: **IMAG.BAS** **(also found in examples directory)**

Description:

The imaginary part of a complex number is returned with **IMAG** and the real part with REAL. To express the parts of a complex number in polar form, use ABS and ARG:

```
PRINT "Rectangular form: Real = ";REAL(Z),"Imag =";IMAG(Z)
PRINT "Polar form: Magnitude = ";ABS(Z),"Angle = ";ARG(Z)
```

See Also:

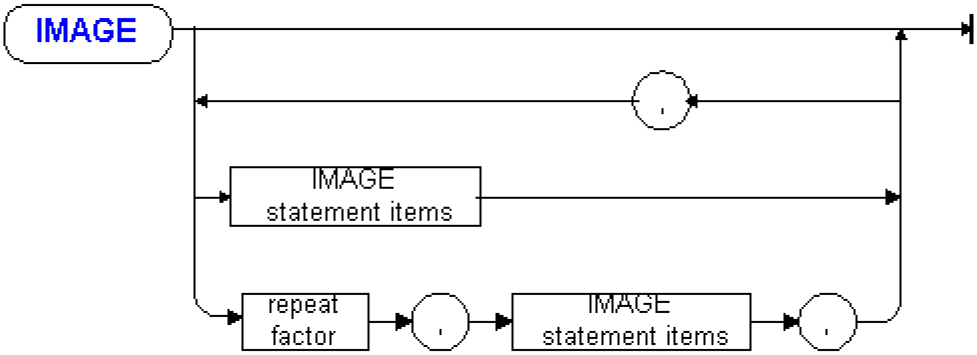
ABS, ARG, CMPLX, CONJG, REAL

IMAGE

Defines the format for data input and output.

Syntax: IMAGE image-specifier [,image-specifier...]

where: image-specifier = # | % | K | -K | H | -H | B | W | Y | + | - |
[repeat-factor] A... | [repeat-factor] X... |
[repeat-factor] /... | [repeat-factor] L... |
[repeat-factor] @... | numeric-specifier |
"string-literal"
numeric-specifier = [S|M] [left-digits] [.|R] [right-digits] [exp]
left-digits = [repeat-factor] {D|Z|*}...
right-digits = [repeat-factor] D...
exp = E | ESZ | ESZZ | ESZZZ
repeat-factor = integer-constant (1 to 32767)



Sample:

```
IMAGE 4ZZ.DD,3X,K,/
PRINT USING """Results = "",SDDDE,3(XX,ZZ)";R,Array(*)
OUTPUT KBD USING "#,B,A"; 255,"K"
ENTER KBD USING 30;X
```

View Sample: **IMAGE.BAS** **(also found in examples directory)**

Description:

Executing an **IMAGE** statement by itself does nothing. The **IMAGE** statement is used to format data for the ENTER, OUTPUT, DISP, LABEL and PRINT USING statements. These statements may use an **IMAGE** statement as their format by specifying the line number or label name of the **IMAGE** statement. Alternately, they can contain a string expression containing the image. To embed quotation marks in a string literal, include two quotation marks.

A complex number is treated like two real numbers and should be specified with two image specifiers. The first specifier defines how the real part should be output/entered and the second specifier does the same for the imaginary part.

The image specifiers in the image list are acted upon as they are encountered. Each specifier should have a matching OUTPUT/ENTER item. Processing of the image list stops when no matching OUTPUT/ENTER item is found. Conversely, the image list is reused starting at the beginning to provide matches for all remaining OUTPUT/ENTER items.

If more decimal places to the left of the decimal point are required to output a numeric item than are specified in the image specifier, an error is generated. If M or S are not specified, then a minus sign will take up one digit place. If the number contains more decimal places to the right of the decimal point than are specified in the image field, the output is rounded to fit.

If the number of characters specified in an image specifier for a string is less than the number of characters in a string, then the remaining characters are ignored. If the number of characters specified is greater than the number of characters in a string then trailing blanks are used to fill out the image field.

OUTPUT, etc.

IMAGE specifiers have the following meanings in DISP, LABEL, OUTPUT and PRINT statements:

#	Suppress automatic output of EOL following the last item.
%	Is ignored in OUTPUT images.
K	Output a number or string in default format, with a period for the radix.
-K	Means the same thing as K.
H	Output a number or string, default format, comma radix.
-H	Means the same thing as H.
B	Output a byte, like the CHR$ function. If the value is larger than 32767, 255 is sent. If the value is smaller than -32768, 0 is sent. If the value is in between, it is rounded to an integer and the least significant byte (CINT(value) MOD 256) is sent.
W	Output a word in 2's complement 16-bit integer form. If the value is larger than 32767, 32767 is sent. If the value is smaller than -32768, -32768 is sent. If the interface is 16-bit, the word is output in one operation (even if the BYTE attribute was used in the I/O path). If the interface is 8-bit, the byte ordering depends on the LSB/MSB attribute of the I/O path. If the destination is a string, native byte ordering is always used (LSB FIRST on a PC, MSB FIRST on a Sun or HP Workstation). If the WORD attribute was specified in the I/O path, a pad byte will be output before the word when necessary to achieve word alignment.
Y	Means the same as W, except that word alignment is not done and the BYTE attribute is not ignored.
+	Change the automatic output of EOL to carriage-return after the last item.
-	Change the automatic output of EOL to line-feed after the last item.
M	Output a minus sign if negative, a blank if positive.
S	Output the sign of the number (+ or -).
D	Output one numeric digit character. The leading zero's are replaced by blanks, a minus sign is displayed on negative numbers.
Z	Means the same thing as D except leading zeros are displayed.
*	Means the same thing as D except leading zeros are replaced with asterisks.
.(period)	Output a decimal-point radix indicator.
R	Output a comma radix indicator.
E	Output an 'E', a sign character and a two-digit exponent.
ESZ	Output an 'E', a sign character and a one-digit exponent.
ESZZ	Output an 'E', a sign character and a two-digit exponent.
ESZZZ	Output an 'E', a sign character and a three-digit exponent.

A	Output an alphanumeric string character.
X	Output a blank.
/	Output a carriage-return and line-feed.
L	Output the current EOL sequence. The default is CR/LF.
@	Output a form-feed character.
"string-literal"	Output the characters in the string literal. Remember to double the quote marks when the image is not in an **IMAGE** statement.

ENTER

IMAGE specifiers have the following meanings in an ENTER statement:

#	Causes the statement to terminate when the last item is terminated. No statement terminator is needed, EOI and LF are item terminators and early termination is not allowed.
%	Is the same as # except EOI causes early statement termination when it terminates an item.
K	Allows free-field entry. For **numerics**, entered characters are sent to the number builder, leading non-numeric characters and blanks are ignored, trailing non-numeric characters and characters sent with EOI true are delimiters. For **strings**, entered characters are sent to the string. A CR may be sent to the string if it is not followed by a LF. The string is terminated by CR/LF, LF, character received with EOI true or the string dimensioned length being filled.
-K	Is like K except LF and CR/LF are not terminators.
H	Is the same as K except a comma is the radix indicator and a period is a non-numeric character.
-H	Means the same as -K for strings and H for numbers.
B	Demands one Byte, like the NUM function.
W	Demands a 16-bit Word (2's complement integer). If the interface is 16-bit, the word is entered in one operation (even if the BYTE attribute was used in the I/O path). If the interface is 8-bit, the byte ordering depends on the LSB/MSB of the I/O path. If the source is a string, native byte ordering is always used (LSB FIRST on a PC, **MSB FIRST** on a Sun or HP Workstation). If the WORD attribute was specified in the I/O path, a pad byte will be entered before the word when necessary to achieve word alignment.
Y	Is the same as W, except that word alignment is not done and the BYTE attribute is not ignored.
+	Indicates an END (EOI) is needed with the last character of the last item to terminate the ENTER statement. LFs are no longer statement terminators, but are still item terminators.
-	Indicates a LF is needed to terminate the ENTER statement. EOI is ignored; other END indicators cause an error.
S	Same meaning as D.

M Same meaning as D.

D Demands one character for each D or repeat count. Non-numerics are consumed while fulfilling the count but also delimit the number. Blanks embedded in the number are ignored.

Z Same meaning as D.

* Same meaning as D.

.(period) Same meaning as D.

R Has the same meaning as D, plus the number builder is instructed to use a comma as the radix indicator and a period as a non-numeric character.

E Is treated the same as 4D.

ESZ Same as 3D.

ESZZ Same as 4D.

ESZZZ Same as 5D.

A Demands one alphanumeric string character.

X Enters a character and discards it.

/ Skips all characters to the next LF. EOI is ignored.

L Ignored in ENTER.

@ Ignored in ENTER.

"string-literal" One character is skipped for each character in the string literal. Remember to double the quote marks when the image is not in an **IMAGE** statement.

Porting Issues

Entering data from a string using

```
ENTER L$ USING "Y"
```

will always use the internal byte ordering of the computer. For PCs and compatibles, the byte ordering is least significant byte (LSB) first. For Sun SPARCstations and HP Workstations, the byte ordering is most significant byte (MSB) first. This limitation applies to ENTER/OUTPUT with strings only. With devices, the byte ordering can be selected in the ASSIGN statement.

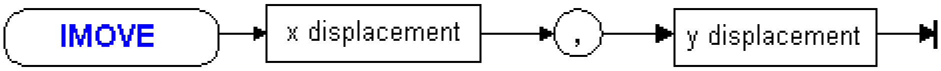
See Also:

ENTER, DISP, LABEL, OUTPUT, PRINT

IMOVE

Lifts and moves the logical pen position incrementally.

Syntax: IMOVE x-displacement, y-displacement



Sample:

```
IMOVE 25,0
IMOVE Xdisp,Ydisp
IMOVE Xx+10,Yy
```

View Sample: **IMOVE.BAS** **(also found in examples directory)**

Description:

The pen is lifted and then moved to the position calculated from adding the specified X and Y displacement to the current pen position. After **IMOVE** executes the logical pen position is updated and the pen is left in the up position.

If you specify a destination which is outside the clipping area, the logical position is set to that point but the pen is not moved.

The PIVOT statement affects the **IMOVE** statement.

See Also:

CLIP, DRAW, IDRAW, IPLOT, LINE TYPE, MOVE, PIVOT, PLOT, RPLOT, SHOW, VIEWPORT, WINDOW

INDENT

Indents a program to reflect its structure.

Syntax: INDENT [start-column [,increment]]

where: start-column = integer-constant in the range 1 to screen-width - 15
increment = integer-constant in the range 0 to screen-width - 15

Sample:

```
INDENT
INDENT 10,5
```

Description:

INDENT is an editing command used to insert spaces after the line numbers and before the leading keywords of a program in order to visually show the structure of the program. The increment value specifies how many spaces to indent each successive structure. The start-column specifies the column to place un-indented lines. The default start-column is seven. The default increment value is two. The **INDENT** statement will move lines starting with REM or a comment tail (!) but will not move comments appended to other statements with a comment tail.

The following statements add a level of indentation: DEF FN, FOR, IF ... THEN, LOOP, REPEAT, SELECT, SUB and WHILE. The following statements are printed one indentation level to the left, but leave the indentation level unchanged: CASE, CASE ELSE, ELSE, EXIT IF, FNEND and SUBEND. The following statements subtract one level of indentation: END IF, END LOOP, END SELECT, END WHILE, NEXT and UNTIL.

This statement can only be executed from the keyboard. It cannot be included in a program.

See Also:

CHANGE, COPYLINES, DEL, DELSUB, EDIT, FIND, MOVELINES, REN, SECURE, XREF

INITIALIZE

Initializes mass storage media.

Syntax: INITIALIZE volume-specifier [,interleave [,option]]

where: interleave and option = numeric-expressions

Sample:

```
INITIALIZE "A:"
INITIALIZE Disc$,2
```

Description:

HTBasic does not support the **INITIALIZE** statement. Use EXECUTE with the “FORMAT” operating system command. Any previous data on the mass storage media is lost when it is initialized. **Be very careful when initializing disks**. It is easy to accidentally initialize the wrong disk, such as a hard disk with hundreds of megabytes of valuable data.

Use either the "FORMAT" command to initialize a disk. For example, use this command to initialize a DOS format floppy disk in drive A:

```
EXECUTE "FORMAT A:"
```

Or use the File Manager to initialize a disk. Select "Disk" and then "Format Disk...". Use your HP Series 200/300 system to initialize a new HP LIF format diskette.

RAM disks are not supported with the **INITIALIZE** ":MEMORY,0" command. Many excellent RAM disk programs are available for the PC that make a RAM disk available to all programs, including HTBasic. These programs can usually make RAM disks in conventional, expanded or extended memory.

See Also:

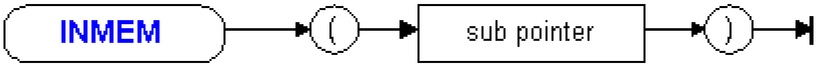
EXECUTE, MASS STORAGE IS

INMEM

Identifies if a subprogram or DLL is loaded.

Syntax: INMEM(sub-pointer)

where: sub-pointer = string expression specifying a subprogram name



Sample: `IF INMEM("Operation") THEN CALL Operation`

```
Present = INMEM("Test")
```

View Sample: **INMEM.BAS** **(also found in examples directory)**

Description:

This function returns one if the specified subprogram has been loaded into memory and zero if it has not. The subprogram must be specified with the initial character in uppercase and subsequent characters in lowercase.

The string expression specifying the subprogram name is called a subprogram pointer because it "points" to the subprogram rather than explicitly naming it. As the expression changes, the pointer points to different subprograms. The following example illustrates how this can be useful.

```
10   SUB Xform(X(*))
20     Method$="Xform"&VAL$(RANK(X))
30     IF NOT INMEM(Method$) THEN LOADSUB Method$
40     CALL Method$ WITH(X(*))
50     DELSUB Method$
60   SUBEND
```

In HTBasic, subprogram pointers can also be used in CALL, DELSUB, LOADSUB and XREF statements.

See Also:

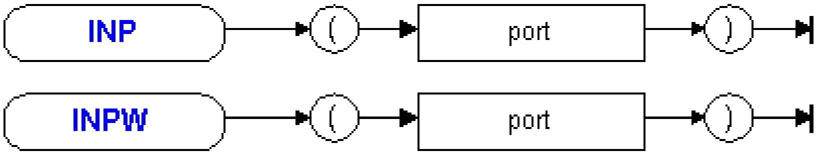
CALL, DELSUB, DLL LOAD, LOADSUB, XREF

INP and INPW

Inputs a byte or word from an I/O Port.

Syntax: INP(port)
INPW(port)

where: port = numeric-expression rounded to an integer



Sample: `PRINT IVAL$(INPW(&H300),16)`
`X=INP(Base+3)`

View Sample: **INP.BAS (also found in examples directory)**
View Sample: **INPW.BAS (also found in examples directory)**

Description:

The **INP** statement inputs a byte from the specified I/O port. The value returned will be an integer in the range 0 to 255. It is equivalent to READIO(8080,Port).

The **INPW** statement inputs an INTEGER from the specified I/O port. It is equivalent to READIO(-8080,Port). These statements are useful for doing I/O with devices, data acquisition boards, etc. for which there is no available device driver.

Some operating systems protect I/O ports; applications are not allowed to read or write them. Under such operating systems, these functions are not allowed. Windows NT is such an operating system.

Porting to HP BASIC:

INP and **INPW** are new HTBasic functions that are not available in HP BASIC. They should not be used in programs that must be ported back to HP BASIC.

See Also:

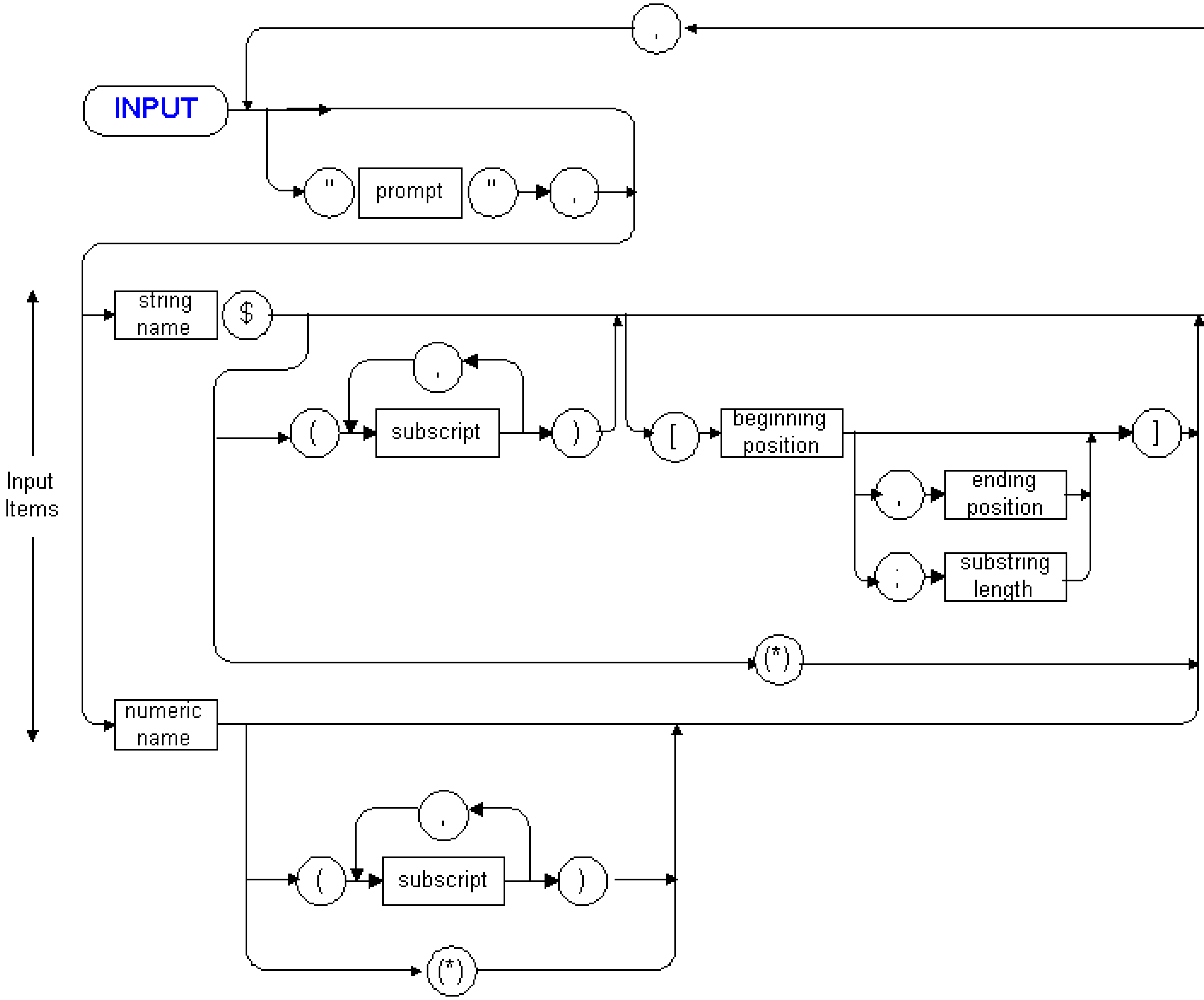
OUT and OUTW, READIO, WRITEIO

INPUT

Inputs numeric or string data from the keyboard.

Syntax: INPUT ["prompt",] item [, ["prompt",] item ...]

where: prompt = string-literal
item = numeric-name [{(subscripts) | (*)}] |
string-name$ [{[(subscripts)] '['sub-string']' |(*)}]
subscripts = subscript [,subscript...]



Sample:
```
INPUT A,B$,C(4),D
INPUT Parray(*)
INPUT "",Str$[1;10]
INPUT "Xcoor=",X,"Ycoor=",Y
INPUT "Enter 4 numbers",Y(1),Y(2),Y(3),Y(4)
```

View Sample: INPUT.BAS **(also found in examples directory)**

Description:

The **INPUT** statement gets information from the user's terminal. The optional prompt string or a question mark (?) is displayed on the CRT display line. The computer then waits until a reply is entered from the keyboard and either CONTINUE or ENTER is pressed to enter a line of input. To suppress the prompt, specify a prompt string of "".

Numeric variables can be simple scalar variables, full array variables, or subscripted array elements. String variables can be simple string variables, array variables, string array elements or sub-strings. An array may be entered in row major order using the full array specifier, "(*)". Complex numbers are entered in rectangular form, first the real part and then the imaginary part.

Leading and trailing spaces are ignored. Data values may be entered individually or multiple values may be entered at once. If multiple values are entered, separate each value with a comma. If too many values are entered, the extra values are ignored. Both quoted and unquoted strings are allowed. Commas are not allowed in unquoted strings, but may appear in quoted strings. To embed one quotation mark in a quoted string, type in two quotation marks at the place you wish one to appear.

Two consecutive commas cause the corresponding variable to retain its old value. Terminating an input line with a comma or pressing CONTINUE or ENTER without entering any data retains the original values for all remaining variables in the list.

Live keyboard operations are not allowed while **INPUT** is waiting for data. ON KBD, ON KEY and ON KNOB events are disabled during **INPUT**.

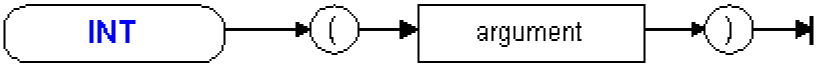
See Also:

DISP, ENTER, LINPUT, OUTPUT, PRINT, READ

INT

Performs the greatest integer function.

Syntax: INT(numeric-expression)



Sample:

```
J4=INT(2.7)
K=INT(-2.7)
Gif=INT(Number)
PRINT "Greatest Integer Function =";INT(Y)
```

View Sample: **INT.BAS** **(also found in examples directory)**

Description:

INT obtains the greatest integer that is less than or equal to the value of its argument. For positive numbers the effect is to truncate the fractional part (if any). For negative numbers, the result is different than you might first expect. For example, the **INT** of 4.9 is 4, but the **INT** of -4.9 is -5 since negative 5 is the largest integer less than negative 4.9.

Notice the differences among CINT, FIX and **INT**. CINT converts a REAL value to an INTEGER value by substituting the closest INTEGER to the value. FIX returns the closest integral value between the REAL value and zero. **INT** returns the closest integral value between the REAL value and negative infinity. Also, CINT actually changes the type from REAL to INTEGER while **INT** and FIX return integral results without changing the type. The following table helps illustrate these differences:

Value x	CINT(x)	FIX(x)	INT(x)
2.6	3	2.0	2.0
2.2	2	2.0	2.0
-2.2	-2	-2.0	-3.0
-2.6	-3	-2.0	-3.0

See Also:

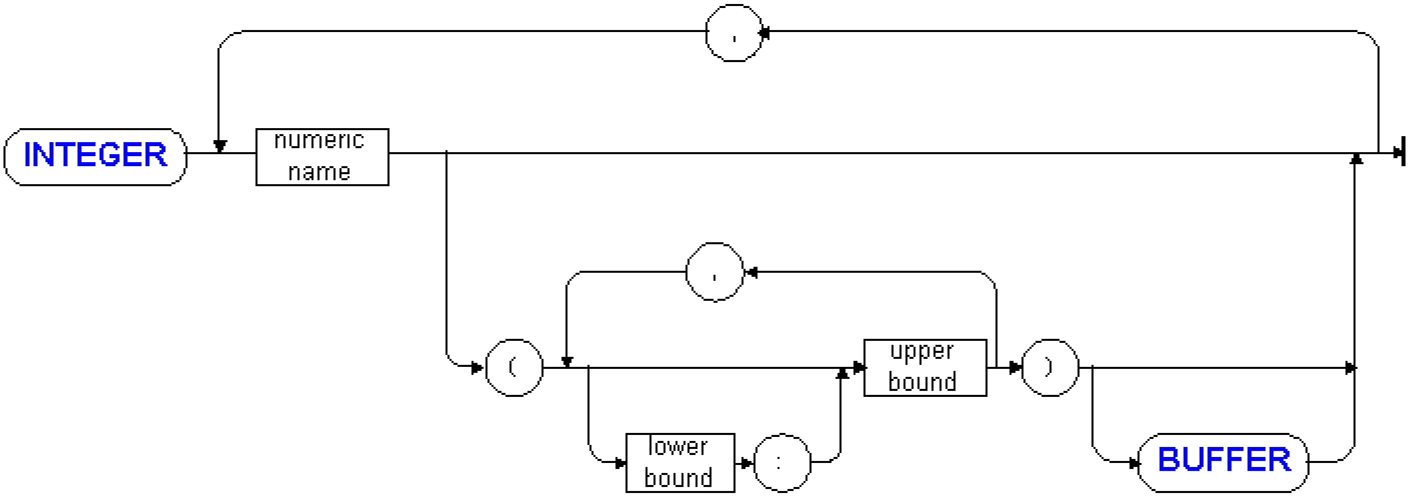
ABS, CINT, DIV, DROUND, FIX, FRACT, MOD, MODULO, PROUND, SGN

INTEGER

Declares and dimensions INTEGER variables.

Syntax: INTEGER item [,item...]

where: item = numeric-name [(bounds) [BUFFER]]
bounds = [lower-bound :] upper-bound [,bounds]
bound = integer constant



Sample:

```
INTEGER I,J,K
INTEGER A,J,Cnt,Point,X(100)
INTEGER Iarray(-128:127,16)
INTEGER Buff(600) BUFFER
```

View Sample: **INTEGER.BAS** **(also found in examples directory)**

Description:

The **INTEGER** statement is used to declare scalar and array variables of type integer. An **INTEGER** variable uses two bytes of storage space. Integer variables conserve memory and integer operations are faster than REAL. REAL is the default type. Bit by bit logical operations may be performed on integer variables.

The maximum number of array dimensions is six and the lower bound must be less than or equal to the upper bound value. Each dimension may contain a maximum of 32,767 elements. An **INTEGER** variable may be declared a buffer by specifying the **BUFFER** keyword after the variable name. Buffer variables are used with the TRANSFER statement.

Any number of **INTEGER** statements are allowed, anywhere in the program; however, an **INTEGER** statement may not appear before an OPTION BASE statement. Memory allocation is made during prerun and cannot be dynamically deallocated. However, the dimensions can be changed in a limited way by REDIM. Use ALLOCATE and DEALLOCATE for dynamic memory allocation.

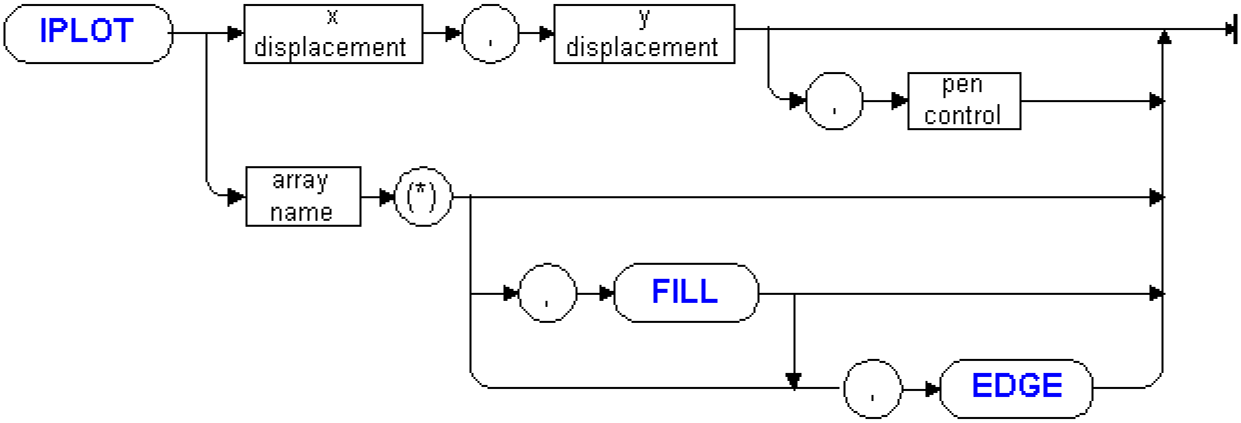
See Also:

ALLOCATE, COM, COMPLEX, DIM, OPTION BASE, REAL, REDIM, TRANSFER

IPLOT

Moves the pen relative to its present location.

Syntax: IPLOT x-displacement, y-displacement [,pen-control]
IPLOT numeric-array(*) [,FILL] [,EDGE]



Sample: `IPLOT 10,0`

```
IPLOT Xdisp,Ydisp,Pen
IPLOT Picto(*),FILL,EDGE
```

View Sample: **IPLOT.BAS** **(also found in examples directory)**

Description:

The **IPLOT** statement moves the pen from its current position by the specified X and Y displacements. The PIVOT and PDIR statements affect the **IPLOT** statement. See PLOT for a full explanation of **IPLOT** arguments.

See Also:

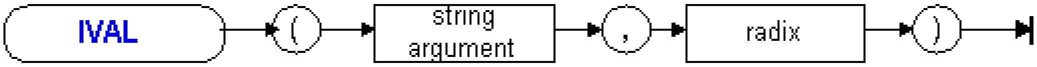
AREA, CLIP, DRAW, IDRAW, IMOVE, MOVE, PLOT, POLYLINE, POLYGON, RPLOT

IVAL

Converts a binary, octal, decimal or hexadecimal string to an INTEGER.

Syntax: IVAL(string-expression, radix)

where: radix = numeric-expression rounded to an integer



Sample:

```
Value=IVAL(Binary$,Two)
PRINT IVAL("FA50",16)
```

View Sample: **IVAL.BAS** **(also found in examples directory)**

Description:

IVAL is like VAL, in that a number in string form is converted to numeric form. Unlike VAL, which can only convert decimal numbers, **IVAL** can convert numbers in binary, octal, decimal and hexadecimal.

The string expression contains the number to be converted and the radix must be either 2, 8, 10 or 16. The characters in the string must be legal digits in the specified radix. For example, a binary number can only have characters "0" and "1". Only decimal numbers are allowed to have a minus sign preceding them.

The number expressed in the string is first converted to a 16 bit integer. If the most significant bit is set, the result will be negative. Thus, the string must represent a number within the range of a 16 bit signed integer. The range restrictions are as follows:

Radix	**Legal Range**
binary	0 through 1111111111111111
octal	0 through 177777
decimal	-32768 through 32767
hexadecimal	0 through FFFF

See Also:

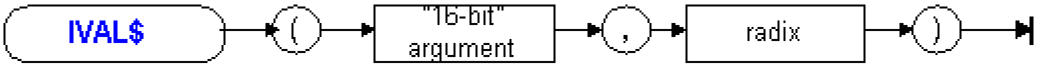
DVAL, DVAL$, IVAL$, VAL, VAL$

IVAL$

Converts an INTEGER to a binary, octal, decimal or hexadecimal string.

Syntax: IVAL$(number, radix)

where: number, radix = numeric-expressions rounded to integers



Sample:

```
Hex$=IVAL$(Number,Sixteen)
PRINT IVAL$(I,8)
```

View Sample: **IVAL$.BAS** **(also found in examples directory)**

Description:

IVAL$** is like VAL$, in that a numeric value is converted to string form. Unlike VAL$, which always expresses numbers in decimal form, **IVAL$ can also express numbers in binary, octal, decimal and hexadecimal.

The number must be in the range -32768 to +32767 and the radix must be either 2, 8, 10 or 16.

The converted numbers have leading zeros as necessary to fill unused digit positions. A minus sign is only produced for decimal numbers. The range of numbers produced is the same as those accepted by IVAL.

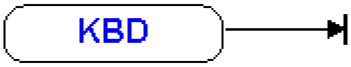
See Also:

IVAL, DVAL, DVAL$, VAL, VAL$

KBD

Returns a 2, the device select code of the keyboard.

Syntax: KBD



Sample: `STATUS KBD;Kbdstat`

```
OUTPUT KBD;Clr$;
```

View Sample: **KBD.BAS** **(also found in examples directory)**

Description:

KBD is an INTEGER function which returns the constant (2), referring to the keyboard interface select code. When referring to the keyboard, **KBD** is more mnemonic than the constant two.

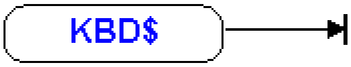
See Also:

CRT, PRT

KBD$

Returns the contents of the ON KBD buffer.

Syntax: KBD$



Sample:

```
PRINT KBD$;
Buff$=Buff$&KBD$
A$=KBD$
```

View Sample: **KBD$.BAS** **(also found in examples directory)**

Description:

When ON KBD is enabled all keystrokes are trapped and held in the keyboard buffer. **KBD$** returns the keyboard contents and then clears it. The buffer is also cleared by the commands: OFF KBD, ENTER KBD, INPUT, LINPUT, SCRATCH and SCRATCH A and by the RESET key. If no key was pressed or if ON KBD is disabled by OFF KBD, the string length is set to zero.

The keyboard buffer can store up to 256 characters. When the buffer is full entering more characters generates a beep and discards the character. Function keys generate 2 bytes. The first byte is 255 and the second byte specifies the function key.

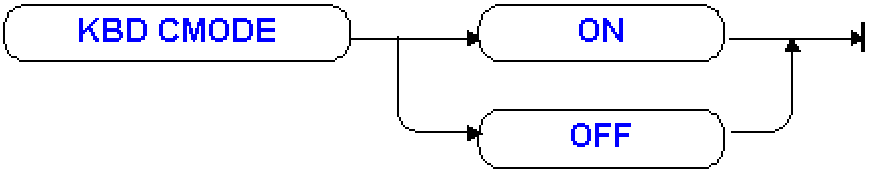
See Also:

OFF KBD, ON KBD

KBD CMODE

Sets softkey compatibility mode.

Syntax: KBD CMODE {ON | OFF}



Sample:

```
KBD CMODE OFF
IF Enable THEN KBD CMODE ON
```

View Sample: **KBD CMODE.BAS** **(also found in examples directory)**

Description:

KBD CMODE controls the softkey emulation mode. HTBasic emulates the ITF keyboard softkeys by default, but can be changed to Nimitz keyboard softkey compatibility mode by using the **KBD CMODE ON** statement. ITF keyboard softkey emulation can be restored by using the **KBD CMODE OFF** statement.

The Nimitz keyboard is used on the 9836 system. It has ten softkeys, and the lowest softkey is labeled k0. The softkey labels are displayed at the bottom of the screen in two rows. Each row contains five labels; each label is 14 characters wide.

See Also:

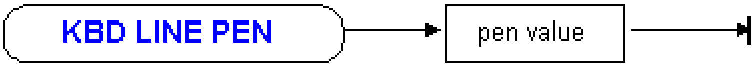
EDIT KEY, KEY LABELS, KEY LABELS PEN, LIST KEY, LOAD KEY, OFF KEY, ON KEY, READ KEY, SCRATCH, SET KEY, STORE KEY, USER KEYS

KBD LINE PEN

Sets the pen color for the input line.

Syntax:

KBD LINE PEN pen-number



Sample:

```
KBD LINE PEN Pen
KBD LINE PEN 141
IF Green THEN KBD LINE PEN Greenpen
```

Description:

This command sets the pen color for the input line, message line, run indicator and edit screen. **KBD LINE PEN** overrides any previous ALPHA PEN for these areas of the screen. The pen-number is a numeric expression rounded to an integer. If you are using the bit-mapped display driver legal values are from 0 to 15. (HP BASIC supports values to 255.) If you are using the non-bit-mapped display driver, legal values are from 136 to 143. This statement is equivalent to CONTROL CRT,17;pen-number.

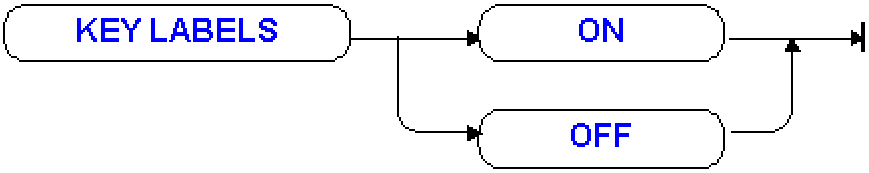
See Also:

ALPHA PEN, KEY LABELS PEN, PRINT PEN

KEY LABELS

Controls the display of the softkey labels.

Syntax: KEY LABELS { ON|OFF }



Sample:

```
KEY LABELS ON
IF Done THEN KEY LABELS OFF
```

View Sample: **KEY LABELS.BAS** **(also found in examples directory)**

Description:

The softkey labels are turned on and off. **KEY LABELS ON** is equivalent to CONTROLCRT,12;2. **KEY LABELS OFF** is equivalent to CONTROLCRT,12;1.

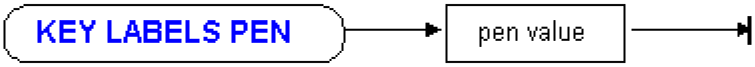
See Also:

EDIT KEY, KBD CMODE, KEY LABELS PEN, LIST KEY, LOAD KEY, OFF KEY, ON KEY, READ KEY, SCRATCH, SET KEY, STORE KEY, USER KEYS

KEY LABELS PEN

Sets the color for the softkey labels.

Syntax: KEY LABELS PEN pen-number



Sample:

```
KEY LABELS PEN Pen
IF Crtb THEN KEY LABELS PEN 4
```

View Sample: **KEY LABELS PEN.BAS** **(also found in examples directory)**

Description:

This statement sets the color for the softkey menu. **KEY LABELS PEN** overrides any previous ALPHA PEN for the color of the softkey menu. The pen-number is a numeric expression rounded to an integer. If you are using the bit-mapped display driver legal values are from 0 to 15. (HP BASIC supports values to 255.) If you are using the non-bit-mapped display driver, legal values are from 136 to 143. This statement is equivalent to CONTROLCRT,16;pen-number.

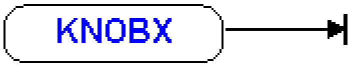
See Also:

ALPHA PEN, KBD LINE PEN, PRINT PEN, OFF KEY, ON KEY, SET KEY

KNOBX

Returns and resets the KNOBX counter value.

Syntax: KNOBX



Sample:

```
Xpulse=KNOBX
IF KNOBX<0 THEN Back
```

View Sample: **KNOBX.BAS** **(also found in examples directory)**

Description:

During an ON KNOB sampling interval, **KNOBX** counts the horizontal mouse pulses generated. Movement of the mouse to the right gives positive counts. Movement in the opposite direction gives negative counts. Once read, the count is cleared. If ON KNOB is not active, **KNOBX** returns a 0.

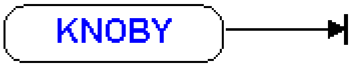
See Also:

KNOBY, ON KNOB

KNOBY

Returns and resets the KNOBY counter value.

Syntax: KNOBY



Sample:

```
Ypulse=KNOBY
IF KNOBY<0 THEN Up
```

View Sample: **KNOBY.BAS** **(also found in examples directory)**

Description:

During an ON KNOB sampling interval, **KNOBY** counts the vertical mouse pulses generated. Upward mouse movement gives positive counts. Movement in the opposite direction gives negative counts. Once read, the count is cleared. If ON KNOB is not active, **KNOBY** returns a 0.

See Also:

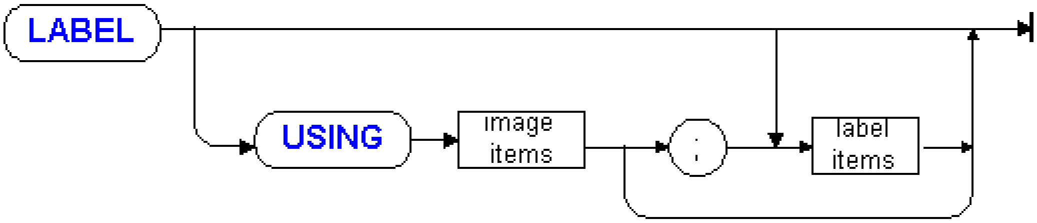
KNOBX, ON KNOB

LABEL

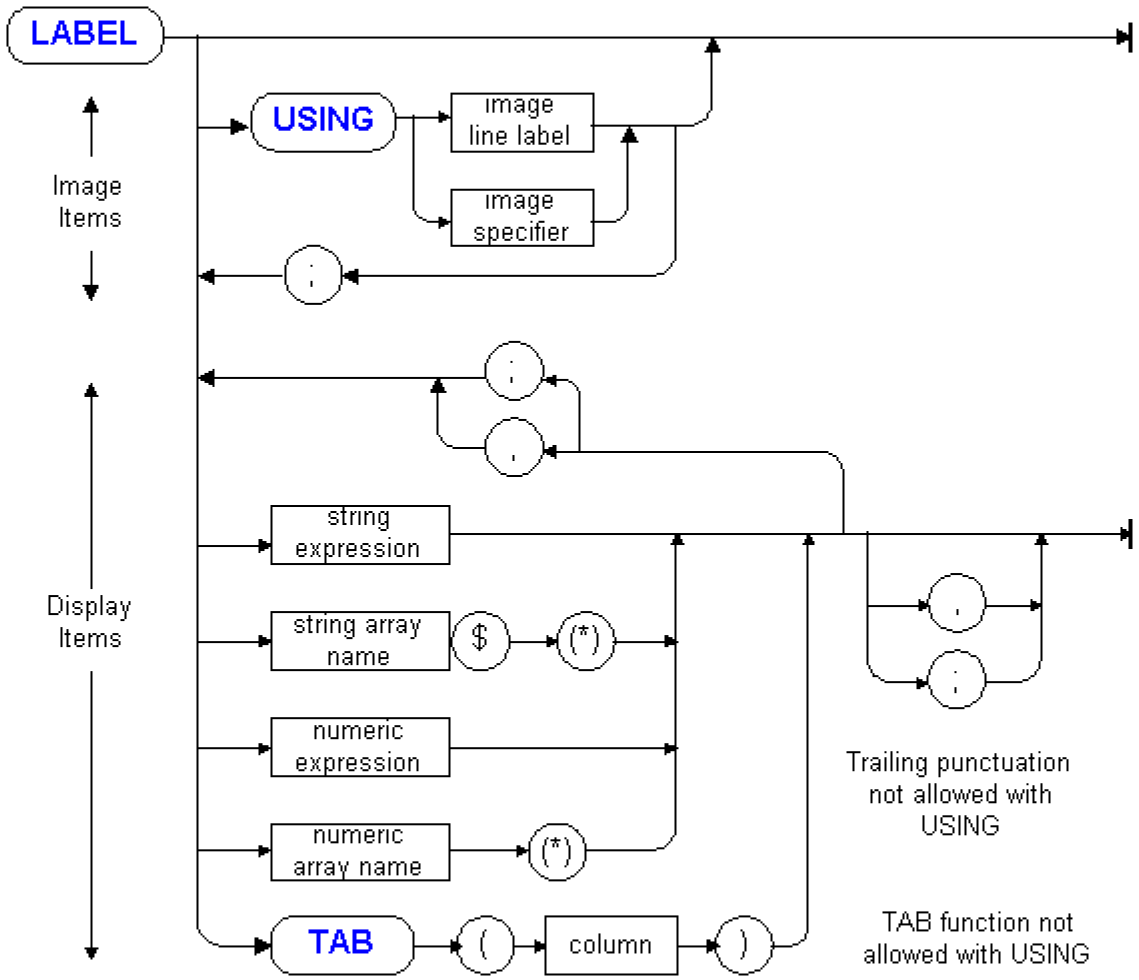
Prints text on graphic devices.

Syntax: LABEL [items [{,|;}]]
LABEL USING image [;items]

where: items = item [{,|;} item [{,|;} item...]]
item = string-expression |
string-array$(*) |
numeric-expression |
numeric-array(*)
image = line-number | line label | string-expression
See IMAGE for image syntax.



expanded diagram:



Sample:

```
LABEL 6,Foobar$
LABEL Array(*)
LABEL USING 160;X,Y,Z
LABEL USING " ""$"",5*.DD";Money
```

View Sample: **LABEL.BAS** **(also found in examples directory)**

Description:

Labels are drawn with the pen beginning at the current pen position, in the current PEN color and LINE TYPE. Labels are clipped at the clip boundary. The starting point for labels is affected by PIVOT, CSIZE, GFONT IS, LORG, and LDIR affect the output of labels, however WINDOW and SHOW do not.

Control Characters

The following control characters have a special meaning when used in **LABEL** statements:

Character	Meaning
CTRL-H, CHR$(8)	moves pen left one character cell.
CTRL-J, CHR$(10)	moves pen down one character cell.
CTRL-M, CHR$(13)	moves pen left length of completed label.

In other respects, the format of output from the **LABEL** statement, both with and without USING, is similar to the PRINT command. See PRINT for an explanation of arrays, numeric and string fields and numeric and string formats.

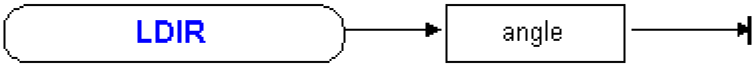
See Also:

CSIZE, GFONT IS, IMAGE, LDIR, LINE TYPE, LORG, PEN, PIVOT, PRINT, SYMBOL

LDIR

Sets the angle for drawing LABELs and SYMBOLs.

Syntax: LDIR angle



Sample: `LDIR 270`

`LDIR ACS(A)`

View Sample: **LDIR.BAS** **(also found in examples directory)**

Description:

The angle is a numeric-expression and is interpreted in the current trigonometric mode, radians or degrees. The default is radians. A value of zero specifies drawing along the positive x-axis. Positive values specify a counter-clockwise direction.

See Also:

CSIZE, DEG, LABEL, LORG, PIVOT, PDIR, RAD, SYMBOL

LEN

Returns the number of characters in a string.

Syntax: LEN(string-expression)

Sample:

```
L=LEN("Four")
IF LEN(A$)=0 THEN Null
```

View Sample: LEN.BAS **(also found in examples directory)**

Description:

The **LEN** function evaluates the string expression and returns the number of characters in the resulting string. If there is nothing in the string, the **LEN** function returns a zero value.

See Also:

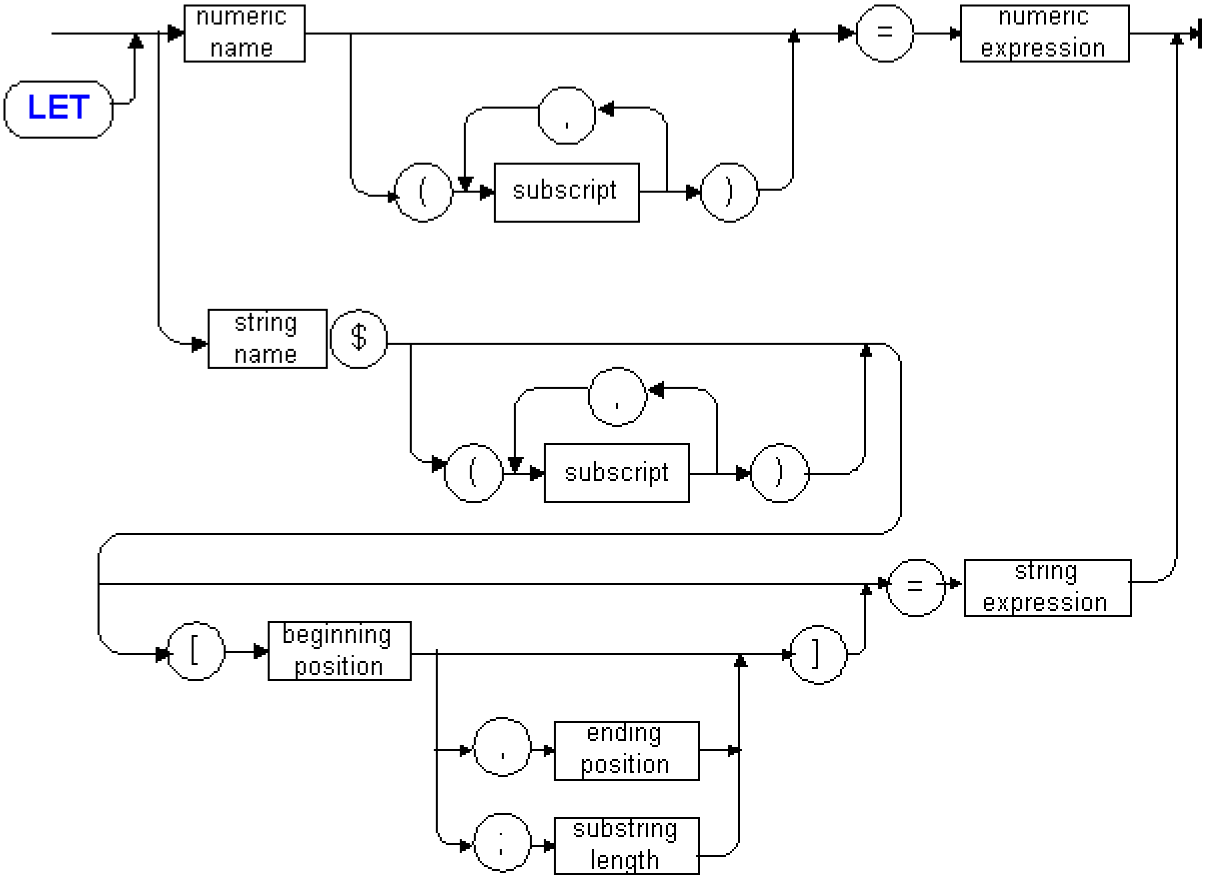
CHR$, LWC$, MAXLEN, NUM, POS, REV$, RPT$, TRIM$, UPC$

LET

Assigns a value to a variable.

Syntax: [LET] numeric-name [(subscripts)] = numeric-expression
[LET] string-name$ [(subscripts)] [sub-string] = string-expression

where: subscripts = subscript [,subscript...]



Sample:

```
LET X=4.2
LET A$="Data Value"
Carray(N+2)=Carray(N)/2
Dat$(5)[1;2]=CHR$(27)&"?"
```

View Sample: LET.BAS **(also found in examples directory)**

Description:

The **LET** keyword is optional. The variable can be a numeric scalar or a numeric array element, a string, a string array element or a sub-string. It can appear on both sides of the equals sign. One assignment is performed in a **LET** statement. Any other equal signs are relational operators in expressions.

If the variable is of type INTEGER, the value of the numeric expression is rounded to an integer. If the value is too large for an INTEGER, an error is generated.

If the string expression length is greater than the dimensioned length of the string, an error is generated. If the assignment is to a sub-string, the string expression length is truncated or blank filled on the right to fit the destination sub-string. If only the sub-string start position is given, the string expression is assigned to the sub-string and the length of the string variable is set.

Use the MAT statement for array assignments.

See Also:

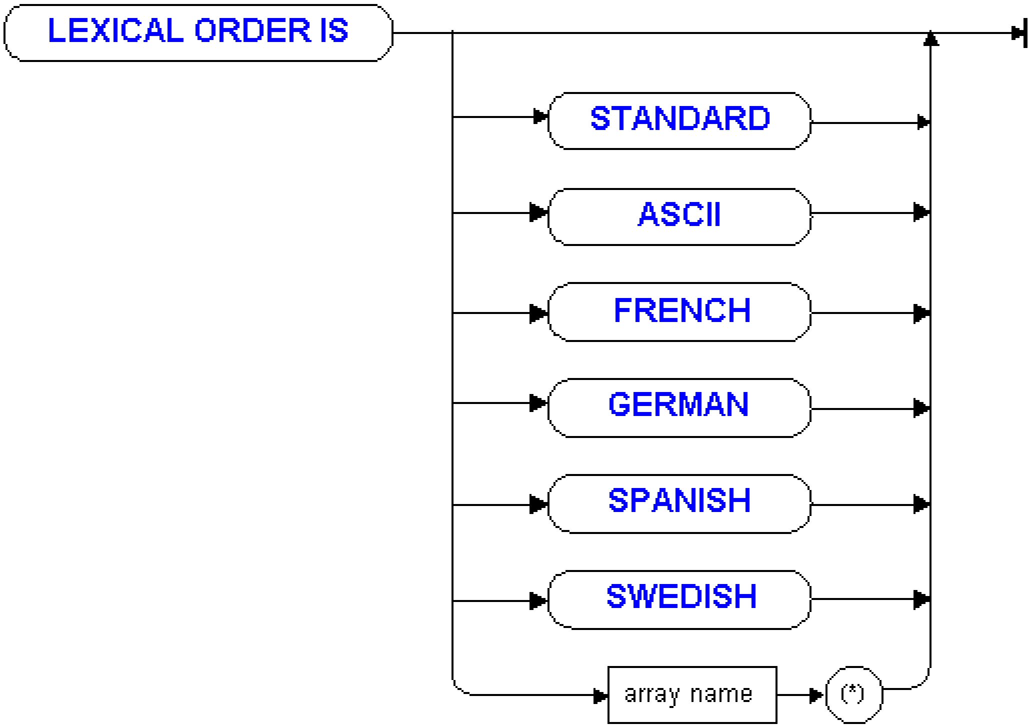
ALLOCATE, COM, DEALLOCATE, DIM, INTEGER, OPTION BASE, REAL

LEXICAL ORDER IS

Defines "alphabetical" order for string comparisons.

Syntax: LEXICAL ORDER IS option

where: option = STANDARD | ASCII | FRENCH | GERMAN | SPANISH | SWEDISH | numeric-array(*)



Sample: `LEXICAL ORDER IS ASCII`
`LEXICAL ORDER IS Mytable(*)`

View Sample: **LEXICAL ORDER IS.BAS** **(also found in examples directory)**

Description:

This statement defines the lexical order of characters to match the alphabets of various languages. The **LEXICAL ORDER IS** statement changes rules for collating order and upper/lower case conversions. Normally, rules for five languages are built into HTBasic: **ASCII, FRENCH, GERMAN, SPANISH,** and **SWEDISH**. (In HTBasic, **LEXICAL ORDER IS STANDARD** is the same as **LEXICAL ORDER IS ASCII**).

The current **LEXICAL ORDER** can be determined with the SYSTEM$("LEXICAL ORDER IS") function.

You may define your own **LEXICAL ORDER** rules using the **LEXICAL ORDER IS** Array(*) syntax. The array is a one dimension INTEGER array of at least 257 elements which contains the rule definitions. The *User's Guide* explains how to set the array elements to the define rules. In addition to collating rules, HTBasic allows you to also specify upper/lower case conversion rules.

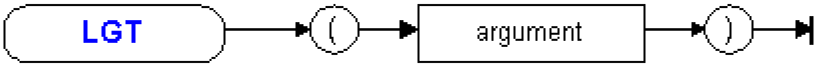
See Also:

LWC$, SYSTEM$, UPC$

LGT

Computes common (base 10) logarithms.

Syntax: LGT(numeric-expression)



Sample:

```
N7=LGT(Xt*4+K)
PRINT "Log of ";Y;"=";LGT(Y)
Db=10*LGT(Watts)
```

View Sample: LGT.BAS **(also found in examples directory)**

Description:

The definition of common or base 10 or Briggsian logarithms is Y = **LGT**(X) where X = 10^Y. **LGT** accepts either a COMPLEX or REAL argument and returns a value of the same type.

COMPLEX Arguments

For COMPLEX arguments **LGT**(Z) is calculated (using complex arithmetic) as

LGT(Z) = LOG(Z)/LOG(10)

The domain of **LGT** includes all points in the complex plane except the origin. However, intermediate values generated during the calculation of the function can cause overflow or underflow errors for very large or small values of Z.

See Also:

EXP, LOG, SQRT

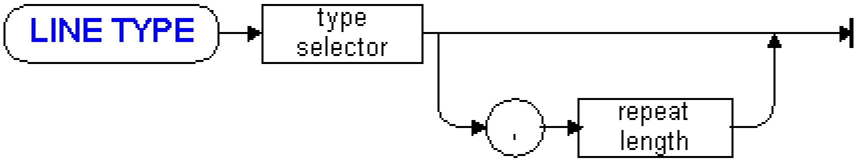
LINE TYPE

Sets the style or dash pattern and repeat length of lines.

Syntax:

LINE TYPE type [,repeat]

where: type and repeat = numeric-expressions, rounded to integers.



Sample: `LINE TYPE 5`

`LINE TYPE Style,Repeat`

View Sample: **LINE TYPE.BAS** **(also found in examples directory)**

Description:

At start-up the default **LINE TYPE** is one (1) for solid lines. When the PLOTTER IS device is not the CRT, the line types are device dependent. Refer to your device documentation. The repeat factor is the GDU line length before the line pattern is repeated.

The CRT line types are:

Value	**Line Type**
1	solid line (default setting)
2	dot at end of line
3	loosely spaced dots
4	closely spaced dots
5	dashes
6	dash, dot
7	large dash, small dash
8	dash, dot, dot
9	solid line, short line at end
10	solid line, long line at end

Under Windows not all line types are supported. Also, most drivers ignore the repeat value.

See Also:

DRAW, IDRAW, IPLOT, PLOT, POLYGON, POLYLINE, RECTANGLE, RPLOT

LINK

Makes a hard link to a file.

Syntax: LINK path1 TO path2 [;PURGE]

where: path1,path2 = file-specifiers

Sample:

```
LINK "/diskless1/htb.hlp" TO "/diskless2/htb.hlp"
LINK Exists$ TO New$;PURGE
```

Description:

Path1 is a file specifier naming an existing file. *Path2* is a file specifier naming a new directory entry to be created. **LINK** automatically creates a new link (directory entry) for the existing file and increments the link count of the file by one. If *path2* already exists, an error is given unless the PURGE option is included.

With hard links, both files must be on the same file system. Both the old and the new link share equal access and rights to the underlying object. The super-user may make multiple links to a directory. Unless the caller is the super-user, the file named by *path1* must not be a directory. LINK_MAX specifies the maximum allowed number of links to the file (see the UNIX man page for pathconf(2V)).

Because a link merely establishes a second name for a single file, operations on that file are effective for all the links to the file. In other words, if the file is changed using one of the filenames, the changes are visible through all the other filenames linked to that file. (Note that this general rule is true in all cases under HTBasic, but is not true under HP BASIC for RE-STORE and RE-SAVE.)

Note: LINK is not supported by HTBasic, it will return an error.

See Also:

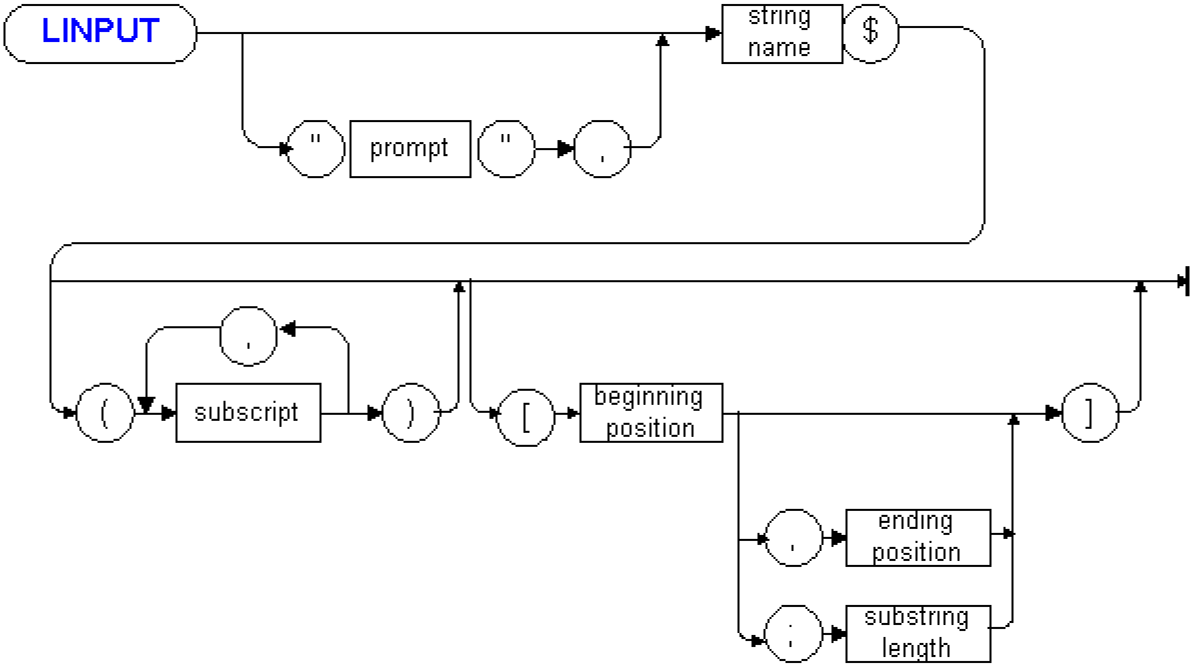
COPY, CREATE, PURGE

LINPUT

Reads alphanumeric keyboard input to a string.

Syntax: LINPUT ["prompt",] string-name$ [(subscripts)] [sub-string]

where: prompt = string-literal
subscripts = subscript [,subscripts]



Sample: `LINPUT "Choice?",D$`
`LINPUT Iarray$(I)[4]`

View Sample: LINPUT.BAS **(also found in examples directory)**

Description:

The **LINPUT** statement gets one alphanumeric data item from the keyboard and assigns it to the string variable. **LINPUT** values may consist of commas, quotation marks and leading and trailing blanks.

The CRT display line will display a prompt while the **LINPUT** is active. If no prompt string is specified a question mark is displayed. If a zero length string-literal is specified, "", the question mark is suppressed. After entry completion, press ENTER.

During an **LINPUT** the ON KBD, ON KEY, and ON KNOB event definitions are deactivated.

See Also:

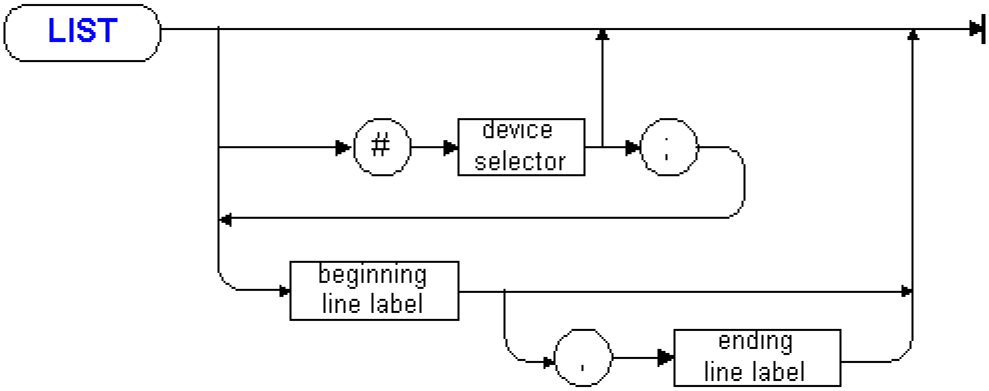
DISP, ENTER, INPUT, OUTPUT, PRINT, READ

LIST

Lists the program in memory to the selected device.

Syntax: LIST [#device-selector [;begin-line [end-line]]]

where: line = line-number | line-label



Sample:

```
LIST
LIST #702
LIST 1500,Endtest
```

View Sample: **LIST.BAS** **(also found in examples directory)**

Description:

The **LIST** statement outputs the program to the PRINTER IS device. If a device selector is given the output is directed to that device. The starting and ending program line numbers may be specified to limit the portion of the program that is output. If the ending line number is not specified, all lines from the start line number through the last line number are output.

After **LIST**ing a program, the available memory in bytes is displayed on the message line.

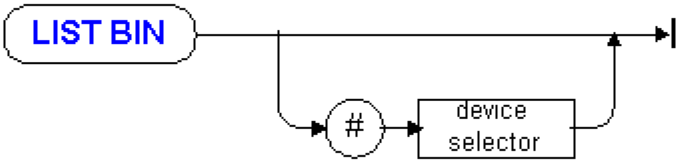
See Also:

GET, LIST BIN, LIST KEY, LOAD, LOADSUB, SAVE, RE-SAVE, STORE, RE-STORE

LIST BIN

Lists each BIN currently in memory.

Syntax: LIST BIN [#device-selector]



Sample: `LIST BIN`

`LIST BIN #PRT`

View Sample: LIST BIN.BAS **(also found in examples directory)**

Description:

BIN files implement HTBasic extensions, such as device drivers. The **LIST BIN** statement prints the name and version number of each **BIN** currently in memory. If a device selector is given, the output is directed to that device, otherwise it is printed on the current PRINTER IS device.

Porting to HP BASIC:

LIST BIN is programmable in HTBasic, but not in HP BASIC.

See Also:

LIST, LIST KEY, LOAD BIN

LIST DLL

Lists the name of each Dynamic Link Library (DLL) currently in memory.

Syntax:

LIST DLL

Sample:

```
LIST DLL
LIST DLL #PRT
```

Description:

The LIST DLL lists the name of each Dynamic Link Library (DLL) function and variable currently in memory.

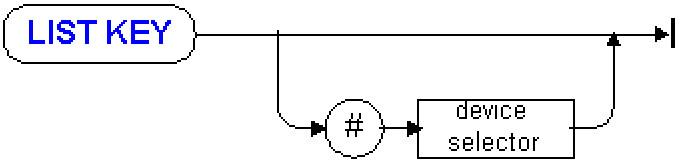
See Also:

DLL GET, DLL LOAD, DLL READ, DLL UNLOAD, DLL WRITE

LIST KEY

Lists the softkey macro definitions.

Syntax: LIST KEY [#device-selector]



Sample: `LIST KEY`

View Sample: **LIST KEY.BAS** **(also found in examples directory)**

Description:

The **LIST KEY** statement outputs the softkey definitions to the PRINTER IS device. If a device selector is given the output is directed to that device. Only defined keys are listed. If the key definition contains an embedded function key then the definition is printed in a special way. The CHR$(255) of the function key is printed as "System Key: ", the 2nd character of the function key is printed and then a new line is started. After all definitions have been printed, the available memory for softkey macros is displayed on the message line.

See Also:

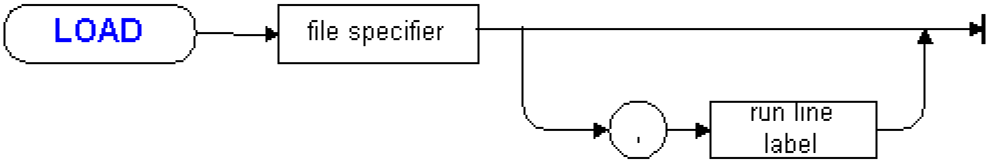
EDIT KEY, KBD CMODE, LOAD KEY, OFF KEY, ON KEY, READ KEY, SCRATCH, SET KEY, STORE KEY, USER KEYS

LOAD

Loads a user program into memory.

Syntax: LOAD file-specifier [,run-line]

where: run-line = line-number | line-label



Sample: `LOAD Story$`
`LOAD "Utility",200`

View Sample: **LOAD.BAS** **(also found in examples directory)**

Description:

LOAD gets a previously stored BASIC program into memory. When **LOAD**ing a program, the current program and all variables not in COM are deleted. Each COM block in the new program is compared to the old COM blocks in memory. Any mismatched or unreferenced COM blocks are deleted. If **LOAD** is used in a program, the newly loaded program begins running at either the first line or the specified line. If **LOAD** is used as a keyboard command and the run line is specified, the program begins running at that line or the next higher line.

PROG files are transportable between different types of computers running HTBasic only if the computers use the same byte ordering. For example, the DOS and Windows versions of HTBasic can share PROG files.

Porting Issues

HTBasic does not support HP BASIC PROG files. To move programs between HTBasic and HP BASIC, **LOAD** the PROG file, SAVE it as an ASCII file, move the program over, GET the ASCII file and STORE it back.

See Also:

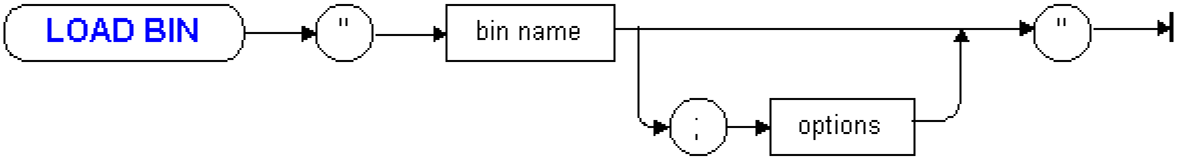
GET, LIST, LOAD BIN, LOAD KEY, LOADSUB, SAVE, RE-SAVE, STORE, RE-STORE

LOAD BIN

Loads a BIN system program file into memory.

Syntax: LOAD BIN "bin-name [;options] "

where: bin-name = file-specifier without extension
options = bin specific option string



Sample: `LOAD BIN "GPIBN;BOARD AT-GPIB"`
`LOAD BIN "SERIAL"`

View Sample: **LOAD BIN.BAS** **(also found in examples directory)**

Description:

The **LOAD BIN** statement loads a BIN system file into memory. BIN files implement HTBasic extensions, such as device drivers. Up to 16 I/O drivers may be loaded. The following three locations are searched for the file, in the order given:

1. The directory specified by the HTB environment variable, if an HTB environment variable exists.
2. The current directory.
3. The directory containing the HTBasic executable.

Some BIN files allow options to be specified. The legal options are different for each device driver; consult the device driver documentation to determine the legal options. Documentation for the standard device drivers included with HTBasic can be found in the *Installing and Using* manual. Documentation for separately available device drivers comes with the driver.

If an error occurs while loading a device driver, it will not be loaded. Often, when an error is detected, more explicit diagnostic information can be obtained by pressing the PRT ALL key to turn print-all mode on (see PRINTALL IS) and retrying the statement **LOAD BIN**.

Under HTBasic, STORE SYSTEM is not an alternative; you must use **LOAD BIN**.

Usage Notes

The search locations for the windows version are:

1. The directory from which the application loaded.
2. The current directory.
3. The Windows system directory (such as \WINNT\SYSTEM32).
4. The Windows directory.
5. The directories listed in the PATH environment variable.

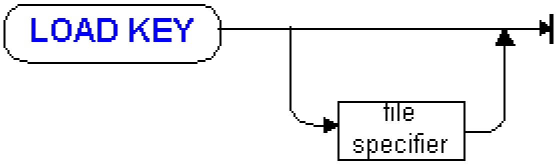
See Also:

LIST BIN, SCRATCH BIN, STORE SYSTEM

LOAD KEY

Loads softkey macro definitions into memory.

Syntax: LOAD KEY [file-specifier]



Sample: `LOAD KEY "DEF"`

View Sample: **LOAD KEY.BAS** **(also found in examples directory)**

Description:

The **LOAD KEY** statement loads softkey macro definitions into memory from a file. Executing **LOAD KEY** without the file specifier resets the softkey definitions to their start-up defaults.

See Also:

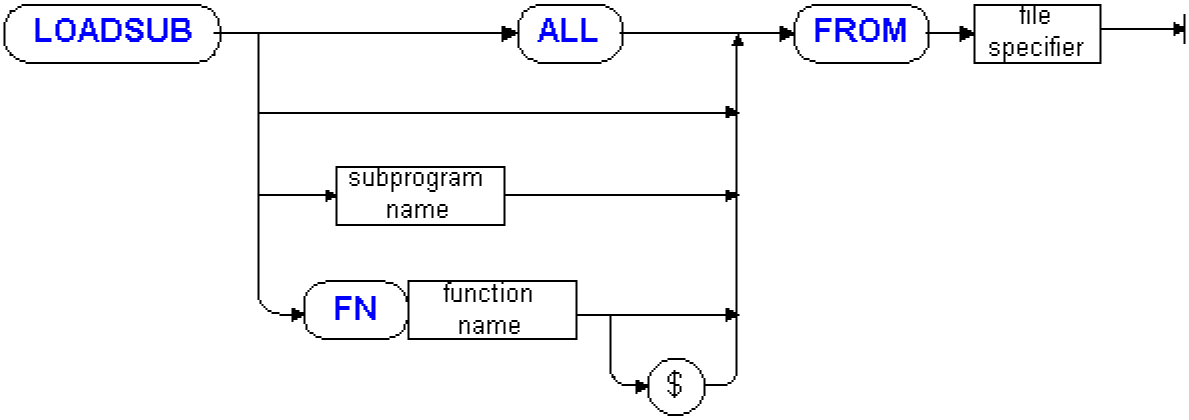
EDIT KEY, KBD CMODE, KEY LABELS, KEY LABELS PEN, LIST KEY, OFF KEY, ON KEY, READ KEY, SCRATCH, SET KEY, STORE KEY, USER KEYS

LOADSUB

Loads a BASIC subprogram into memory.

Syntax: LOADSUB [context] FROM file-specifier

where: context = ALL | subprogram-name | FN function-name[$] | string-expression



Sample:

```
LOADSUB Peek FROM "PEEK.COM"
LOADSUB FROM "Testfile"
LOADSUB FNSearch$ FROM "Sarfile"
LOADSUB ALL FROM Myfile$
LOADSUB Subptr$ FROM "ROUTINES.LIB"
```

View Sample: **LOADSUB.BAS** **(also found in examples directory)**

Description:

The **LOADSUB** statement loads subprograms at the end of the current program. It re-numbers the incoming subprogram lines. After loading a subprogram it also preruns the subprogram to check for COM block mismatches.

If **ALL** is specified, all subprograms in the file are loaded into memory. If a subprogram name is specified (either explicitly or in a string expression), only that subprogram is loaded into memory. These forms of **LOADSUB** are programmable.

LOADSUB FROM (no context specified) looks through a program and loads all subprogram references not yet in memory. The newly loaded subprograms are also looked through and any additional subprogram references not yet in memory are located and loaded into memory. After **LOADSUB FROM** has executed, if any subprogram references were not loaded into memory, an error is generated along with a listing of the subprogram names. **LOADSUB FROM** is not programmable.

Subprogram Pointer

If a string expression specifies the subprogram name in the **LOADSUB** statement, the string expression is called a subprogram pointer because it "points" to the subprogram rather than explicitly naming it. As the expression changes, the pointer points to different subprograms. The following example illustrates how this can be useful.

```
10   SUB Xform(X(*))
20     Method$="Xform"&VAL$(RANK(X))
30     IF NOT INMEM(Method$) THEN LOADSUB Method$
40     CALL Method$ WITH(X(*))
50     DELSUB Method$
60   SUBEND
```

The subprogram pointer must be specified with the initial character in uppercase and subsequent characters in lowercase. Subprogram pointers can also be used in CALL, DELSUB, INMEM, and XREF statements.

Porting to HP BASIC:

The use of subprogram pointers in **LOADSUB** is a new HTBasic feature that is not available in HP BASIC. It should not be used in programs that must be ported back to HP BASIC.

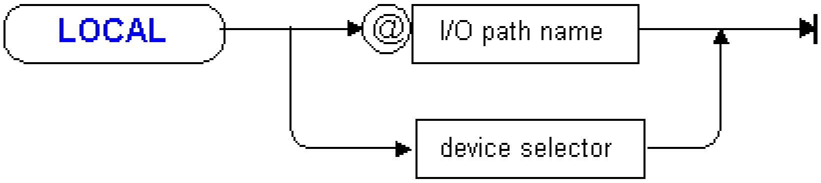
See Also:

CALL, DELSUB, INMEM, RE-STORE, STORE

LOCAL

Returns specified IEEE-488 devices to their local state.

Syntax: LOCAL {@io-path | device-selector}



Sample: `LOCAL @Dvm`

```
LOCAL Isc
LOCAL 728
```

Description:

If a primary device address is specified, a Go To Local (GTL) message is sent to all listeners and LOCAL LOCKOUT is not canceled. If only an interface select code is specified, all devices on the bus are returned to the local state and LOCAL LOCKOUT is canceled.

If a primary device address is specified and the computer is the Active Controller, the bus activity is: ATN, MTA, UNL, LAG, GTL.

If the computer is not the Active Controller but is the System Controller and just an interface select code is specified, the REN line is set false. If it is also the Active Controller the ATN and REN lines are both set false.

When the computer is not the System Controller but is the active controller, the bus activity for an Interface Select Code is to set the ATN line and send a GTL message.

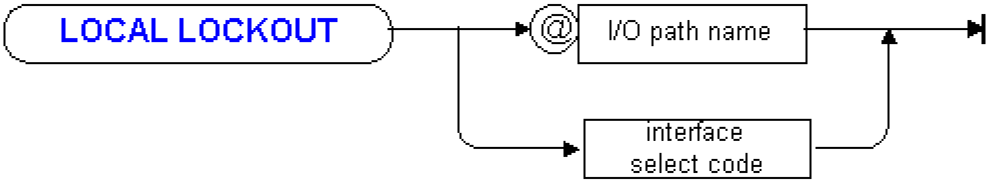
See Also:

ABORT , CLEAR, PASS CONTROL, PPOLL, REMOTE, REQUEST, SEND, SPOLL, TRIGGER

LOCAL LOCKOUT

Sends the IEEE-488 LLO message.

Syntax: LOCAL LOCKOUT {@io-path | interface-select-code}



Sample: `LOCAL LOCKOUT 7`

```
LOCAL LOCKOUT Isc
LOCAL LOCKOUT @Gpib
```

Description:

The local lockout message LLO is sent over the IEEE-488 preventing front panel control of devices in the remote state.

If the computer is not the active controller or a primary device address is specified, an error is generated. If an I/O path is specified, it must refer to the IEEE-488 interface.

See Also:

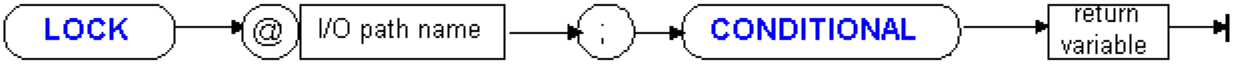
ABORT, CLEAR, LOCAL, PASS CONTROL, PPOLL, REMOTE, REQUEST, SEND, SPOLL, TRIGGER

LOCK

Secures a file for exclusive access.

Syntax: LOCK @io-path; CONDITIONAL return

where: io-path = name assigned to a file
return = numeric-name



Sample:

```
LOCK @Proprietary;CONDITIONAL Result
IF Secure THEN LOCK @Keyfile;CONDITIONAL Ok
```

View Sample: LOCK.BAS **(also found in examples directory)**

Description:

This command attempts to **LOCK** a file to prevent other users from accessing the file while you are using it. The return value is zero if the file is successfully **LOCK**ed and non-zero if the **LOCK** fails. The value returned is an error number, indicating why the **LOCK** failed. An ASSIGN @Path **TO** * will UNLOCK and then close the file.

File locking capabilities depend on the operating system HTBasic is running on. If the operating system does not support it, the result value will always indicate failure. Some operating systems require the **LOCK** request when the file is opened. On such a system, the file will be closed and re-opened with the **LOCK**.

A file can have multiple locks on it. The file remains locked until a corresponding number of UNLOCK statements have been executed. **LOCK**ing a file should be a temporary action of short duration so that fair access to the file is provided to all network users.

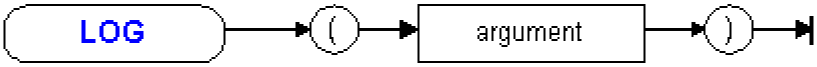
See Also:

ASSIGN, UNLOCK

LOG

Computes natural (base "e") logarithms.

Syntax: LOG(numeric-expression)



Sample:

```
LN=LOG(Dt4)
PRINT "LN(";X;") =";LOG(X)
```

View Sample: **LOG.BAS** **(also found in examples directory)**

Description:

The definition of natural or base "e" or Naperian logarithms is Y = **LOG**(X), where X = EXP(Y). "e" is an irrational number whose value is approximately 2.718 281 828 459 05.

COMPLEX Arguments

LOG accepts either a COMPLEX or REAL argument and returns a value of the same type. For COMPLEX arguments the real and imaginary parts of **LOG**(Z) are calculated (using real arithmetic) as

REAL(LOG(Z)) = LOG(ABS(Z))
IMAG(LOG(Z)) = ARG(Z)

which returns an imaginary part in the range -PI to PI, regardless of the current trigonometric mode. The domain of **LOG** includes all points in the complex plane except the origin. However, intermediate values generated during the calculation of the function can cause over or underflow errors for very large or small values of Z.

See Also:

EXP, LGT, SQRT

LONG

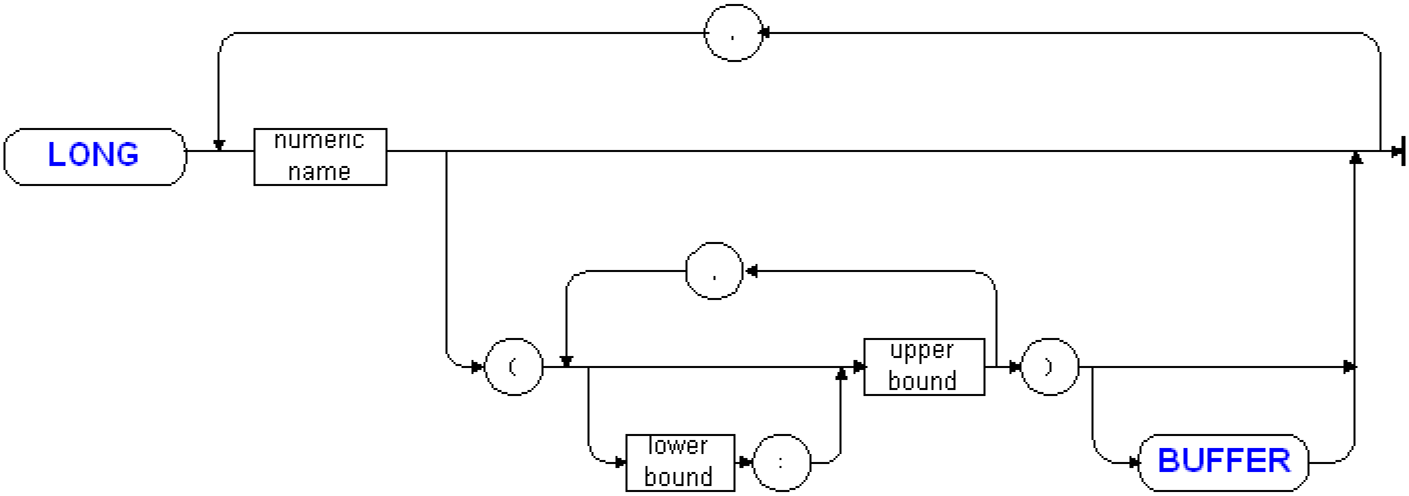
Declares, dimensions and reserves memory for Long integers between -2,147,483,648 and 2,147,483,647.

Syntax:

LONG item [,item...]

where:

item = numeric-name [(bounds) [BUFFER]]
bounds = [lower-bound :] upper-bound [,bounds]
bound = integer constant



Sample:

```
LONG I,J,K
LONG A,J,Cnt,Point,X(100)
LONG Aarray(-128:127,16)
LONG Buff(600) BUFFER
```

Description:

LONG is a data type. Other data types are I/O path, integer, real, complex, and string. LONG declares, dimensions and reserves memory for integers between -2,147,483,648 and 2,147,483,647. LONG variables can be declared and used in exactly the same ways that INTEGER variables are used. The only difference is the range of values allowed.

See Also:

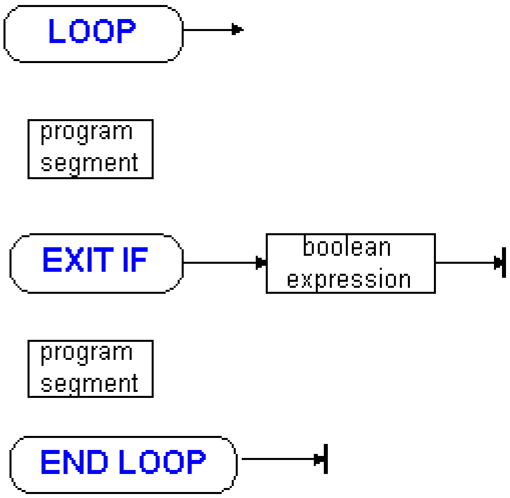
COMPLEX, INTEGER, REAL

LOOP

Defines a series of statements to be executed repeatedly.

Syntax: LOOP
statements
[EXIT IF boolean-expression]
statements
END LOOP

where: statements = zero, one or more program statements



Sample:

```
100  LOOP
. . .
170  EXIT IF J=5 OR A$>B$
. . .
180  END LOOP
```

View Sample: LOOP.BAS **(also found in examples directory)**

Description:

When control reaches the **END LOOP** statement, it is transferred back to the statement following the **LOOP** statement until an **EXIT IF** statement evaluates non-zero. There may be any number of **EXIT IF** statements in the **LOOP**. Branching into a **LOOP** is legal.

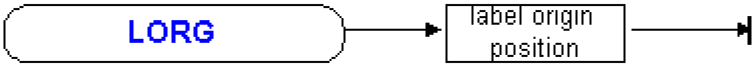
See Also:

CALL, END, FN, FOR, GOTO, GOSUB, IF, ON, PAUSE, REPEAT, RETURN, RUN, SELECT, STOP, SUBEND, SUBEXIT, WAIT, WHILE

LORG

Specifies the position of a LABEL relative to the current position.

Syntax: LORG numeric-expression



Sample:

```
LORG Origin
LORG 2
```

View Sample: **LORG.BAS** **(also found in examples directory)**

Description:

The **LORG** statement specifies the relative position of the LABEL with respect to the current pen position. The argument is rounded to an integer and has a range of one through nine. The default **LORG** origin is one. The values are as follows:

Left Values	**Middle Values**	**Right Values**
3 - left-top	6 - middle-top	9 - right-top
2 - left-center	5 - middle-center	8 - right-center
1 - left-bottom	4 - middle-bottom	7 - right-bottom

If the string length is odd, the horizontal center of the string is the center of the middle character.

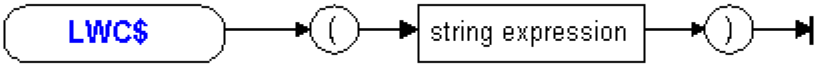
See Also:

CSIZE, IMAGE, LABEL, LDIR, LINE TYPE, PDIR, PEN, PIVOT, PRINT, SYMBOL

LWC$

Converts characters in a string to lowercase.

Syntax: LWC$(string-expression)



Sample: A$=LWC(B$)

```
PRINT LWC$(Answer$)
```

View Sample: **LWC$.BAS** **(also found in examples directory)**

Description:

The upper-case to lower-case correspondence is affected by LEXICAL ORDER IS. If a user-defined table is used with LEXICAL ORDER IS and the optional upper and lowercase conversion rules are not specified, the uppercase to lowercase transform is determined by the STANDARD lexical order.

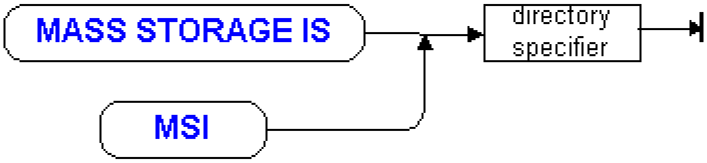
See Also:

CHR$, LEN, LEXICAL ORDER IS, MAXLEN, NUM, POS, REV$, RPT$, TRIM$, UPC$, VAL, VAL$

MASS STORAGE IS

Assigns the current mass storage device and directory.

Syntax: MASS STORAGE IS path-specifier
MSI path-specifier



Sample:

```
MASS STORAGE IS Volspec$&Dir_path$
MSI "A:\DIR1\DIR2\MYDIR"
MSI "/usr/bin" CD "/usr/bin"
```

View Sample: **MASS STORAGE IS.BAS** **(also found in examples directory)**

Description:

The current **MASS STORAGE IS** includes both the device, and the current directory. This current directory is searched first to find any specified files. You may change the current device and directory with the MSI command. You may determine the current device and directory with the SYSTEM$("MSI") function.

MASS STORAGE IS may be abbreviated MSI or CD.

See Also:

CAT, CONFIGURE MSI, COPY, CREATE, INITIALIZE, PRINT LABEL, PROTECT, PURGE, READ LABEL, RENAME, SYSTEM$("MSI")

MAT

Specifies an array operation.

Syntax:

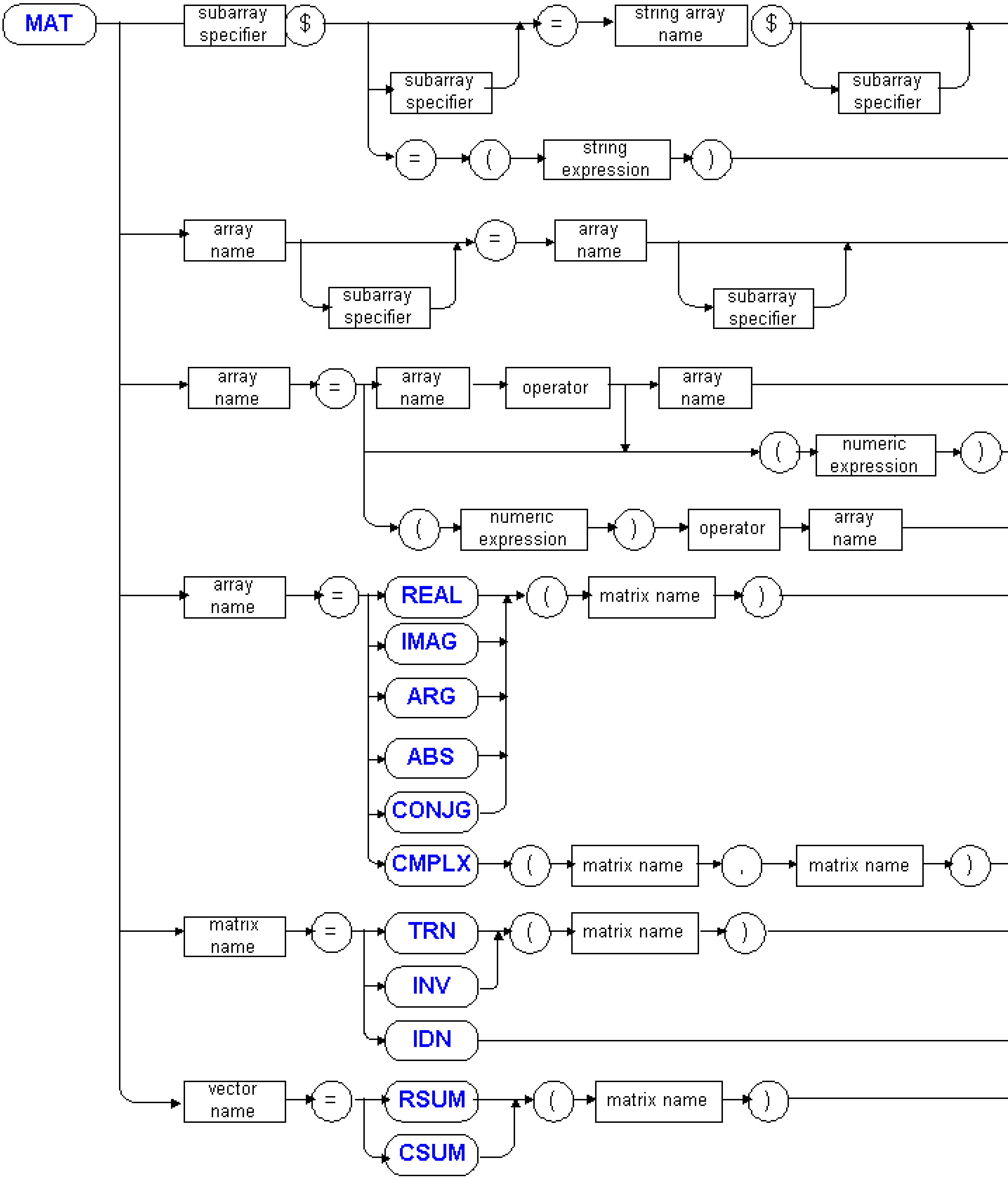
```
MAT string-array$ = string-array$ | (string-expression)
MAT numeric-array = numeric-array [operator numeric-array]
MAT numeric-array = (numeric-expression) [operator numeric-array]
MAT numeric-array = numeric-array operator (numeric-expression)
MAT vector = RSUM(matrix) | CSUM(matrix)
MAT matrix = INV(matrix) | TRN(matrix) | IDN
MAT array-name [sub-array] = array-name [sub-array]
```

COMPLEX Extensions:

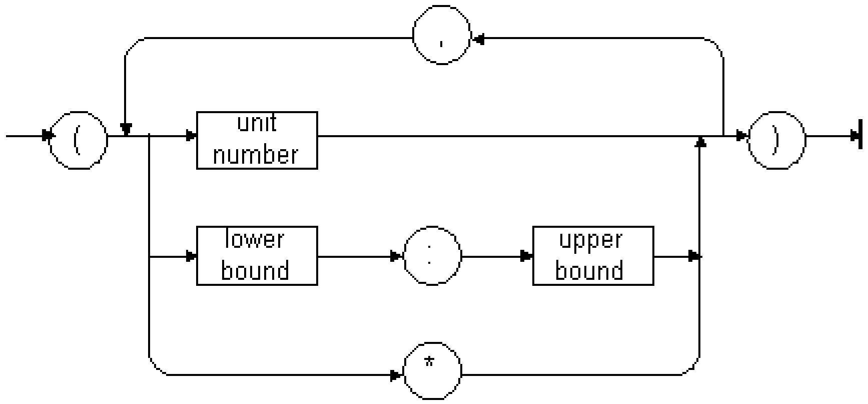
```
MAT array-name = REAL(array-name)
MAT array-name = IMAG(array-name)
MAT array-name = ARG(array-name)
MAT array-name = ABS(array-name)
MAT array-name = CONJG(array-name)
MAT array-name = CMPLX(array-name,array-name)
```

where:

```
operator = + | - | . | / | < | <= | = | <> | >= | > | *
sub-array = ( {range | subscript} [, {range | subscript}...] )
range = * | lower-bound : upper-bound
```


MAT
subarray specifier
$
=
string array name
$
subarray specifier
subarray specifier
=
(
string expression
)
array name
=
array name
subarray specifier
subarray specifier
array name
=
array name
operator
array name
(
numeric expression
)
(
numeric expression
)
operator
array name
array name
=
REAL
(
matrix name
)
IMAG
ARG
ABS
CONJG
CMPLX
(
matrix name
,
matrix name
)
matrix name
=
TRN
(
matrix name
)
INV
IDN
vector name
=
RSUM
(
matrix name
)
CSUM

subarray specifier:



Sample: `MAT A=A*(Pny*6)`

```
MAT A=B+C
MAT A=C>=(1)
MAT A=(4)
MAT A=CSUM(C)
MAT A=RSUM(D)
MAT A=IDN
MAT A=INV(B)
MAT Destination(3,*,*)=Source(*,2,*)
```

View Sample: MAT.BAS **(also found in examples directory)**

Description:

MAT initializes and performs operations on string and numeric arrays. **MAT** operations can copy a string or numeric expression or array into an array, add or subtract an array or numeric expression to an array or numeric expression, multiply or divide an array or numeric expression by an array or numeric expression, compare arrays and numeric expressions or perform an identity (**IDN**), inverse (**INV**), sum (**CSUM** or **RSUM**) or transpose (**TRN**) of rows and columns of a matrix. **MAT** operations can also be used to assign a sub-array to another array or sub-array.

The REAL, IMAG, ARG, ABS, CONJG and CMPLX functions operate the same with arrays as with scalar numbers.

Size and Shape Requirements

In general, a matrix must meet certain size and shape requirements for each matrix operation. If it does not, in certain operations it makes sense to automatically redimension it. If it can't be redimensioned, an error is given.

Sub-array assignments require that the number of ranges specified in the source match the number of ranges specified in the destination. If a complete array is specified, the number of ranges equals the rank of the array. In corresponding ranges of the source and destination, the number of elements must be the same. The following examples will help you visualize these rules:

```
10 DIM X(1:3),Y(1:10)
20 DIM D(3,4,5),S(4,2,5)
30 MAT X=Y(2:4)          ! One range, three elements
40 MAT D(3,*,*)=S(*,2,*) ! Range 1 has 5 elements,2 has 6
50 MAT Y(1:6)=S(0,0,*)   ! One range, 6 elements
```

For the list of operators above, the target array must be the same size and shape as the source array because numeric operations are performed one array element at a time and the result is returned to the corresponding element in the target array.

Matrix Multiply

The asterisk "*" operator performs a matrix multiplication when it is between two matrices. If it is between an array and a numeric expression each element of the array is multiplied by the value of the expression. The period "." operator is used between two arrays to perform an element by element multiply. Vectors can be used in a matrix multiplication as if they were two-dimensional matrices. If used as the first matrix, a vector is treated as a 1 by N matrix. If used as the second matrix, a vector is treated as an N by 1 matrix.

Sum Columns, Rows

The **CSUM** and **RSUM** matrix functions sum the columns and rows, respectively, of a matrix and return the result into a target vector array.

Identity

The **IDN** matrix function initializes a square matrix to an identity matrix. An identity matrix has zeros in all elements but the diagonal elements, which have the value one.

Invert

The **INV** matrix function returns the inverse of a square matrix. It also calculates the DET value. If the matrix has no inverse, the DET is set to zero, but no error is returned. If the DET is very small in relation to values of the array, numerical methods for inverting the array fail. Thus, the DET should be checked after using **INV**.

Transpose

The **TRN** matrix function returns the transpose of the source matrix by exchanging rows for columns and columns for rows.

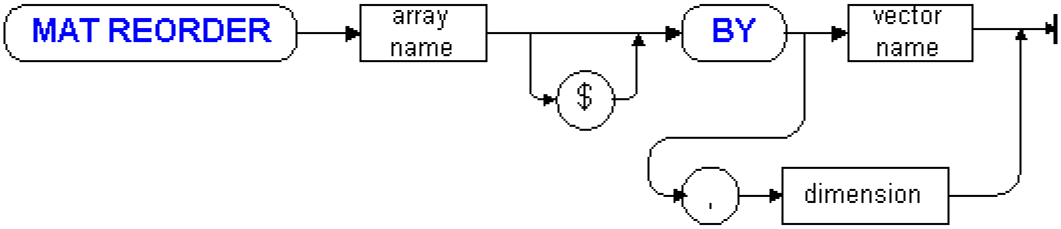
See Also:

DET, DIM, DOT, MAT REORDER, MAT SEARCH, MAT SORT, REDIM, SUM

MAT REORDER

Reorders array elements by a supplied subscript list.

Syntax: MAT REORDER array-name[$] BY vector [, subscript]



Sample: `MAT REORDER Array BY Vector,2`

```
MAT REORDER Elements$ BY New
```

View Sample: **MAT REORDER.BAS** **(also found in examples directory)**

Description:

The array is reordered according to the values in the vector. The optional subscript is rounded to an integer and specifies which subscript is to be reordered. If it is not specified it is assumed to be one.

The vector must be a one dimensional array which is the same size as the specified subscript. It contains integers specifying valid subscript values with no duplicate values. The MAT SORT statement may be used to generate vector values.

COMPLEX Arrays

MAT REORDER can reorder a complex array, but a reorder vector can not be complex.

See Also:

MAT, MAT SEARCH, MAT SORT, REDIM

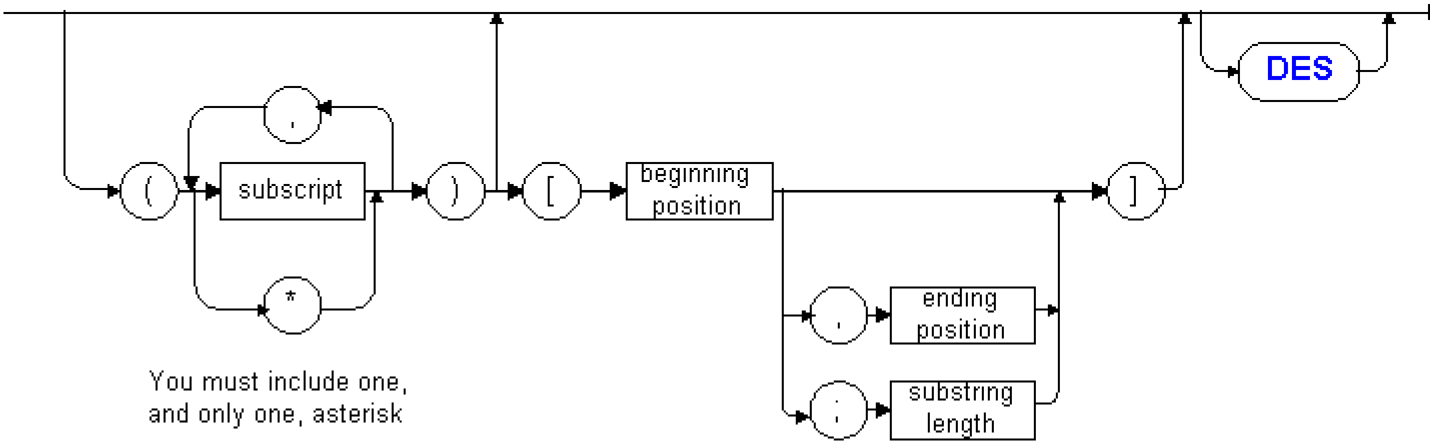
MAT SEARCH

Searches an array for user specified conditions.

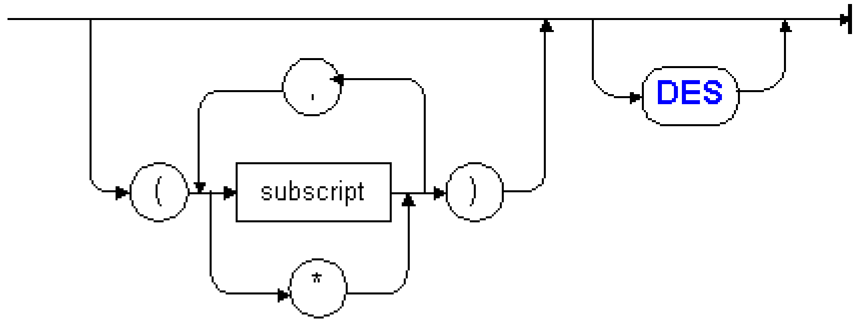
Syntax: MAT SEARCH numeric-array [num-key], rule; return [,start]
MAT SEARCH string-array$ [str-key], rule; return [,start]

where: num-key = [search-subscripts] [DES]
str-key = [search-subscripts [sub-string]] [DES]
search-subscripts = ({subscript|*} [,...])
The '*' must appear only once.
rule = [#]LOC ([relational] value) | LOC MAX | LOC MIN | MIN | MAX
relational = < | <= | = | <> | => | >
return = variable-name
start = numeric-expression
value = string-or-numeric-expression

string key specifier:

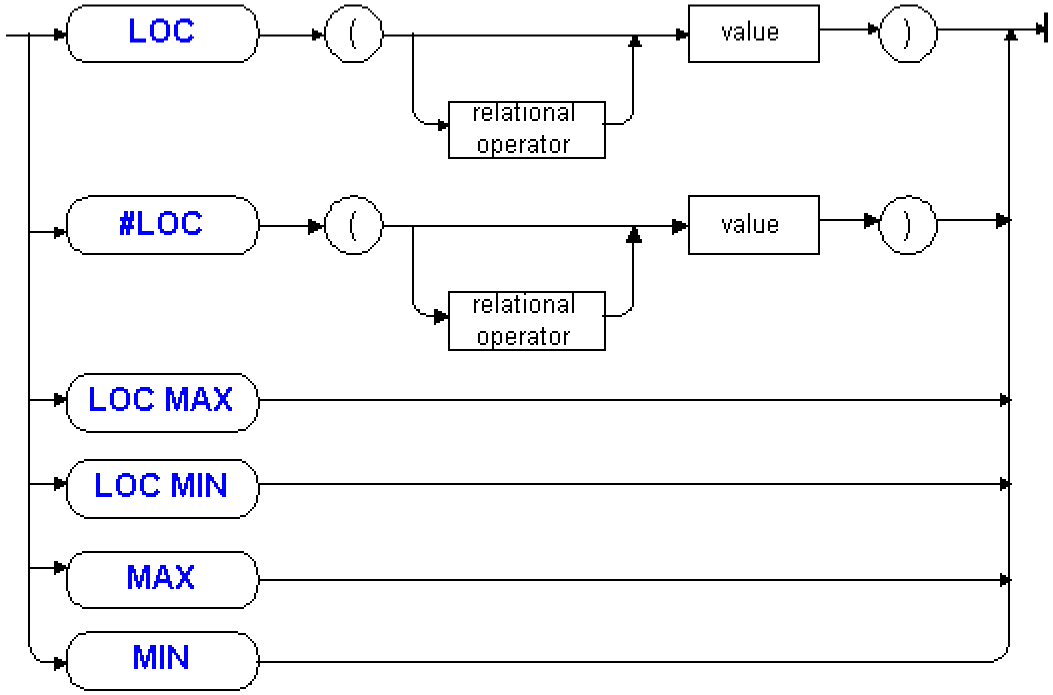


numeric key specifier:



You must include one,
and only one, asterisk

condition specifier:



Sample:

```
MAT SEARCH Vector,#LOC(<>PI);Not_pi
MAT SEARCH Temperature,LOC MAX;Hottest
MAT SEARCH Students,LOC(<.33);Flunk,4
MAT SEARCH Titles$(*,2,3) DES,MAX;Last_book$
MAT SEARCH Array$(*), LOC(=Target$);I
```

View Sample: MAT SEARCH.BAS **(also found in examples directory)**

Description:

A numeric or string array is searched for the specified condition and the result is returned in the return variable. The keyword **DES** specifies descending search order. The optional start value specifies the starting subscript. If not specified, searching begins with the first element for ascending searches and the last element for descending searches. The "rule" specifies the search rules to use and what to return:

Rule	**Meaning**
LOC	Subscript of first element satisfying operator
#LOC	Count the number of elements satisfying operator
LOC MAX	Subscript of maximum value
LOC MIN	Subscript of minimum value
MAX	Find and return the maximum value
MIN	Find and return the minimum value

COMPLEX Arrays

MAT SEARCH can search an array, but since the concept of linear ordering does not apply to the complex plane, greater than, less than, **MIN** and **MAX** operations are not allowed.

See Also:

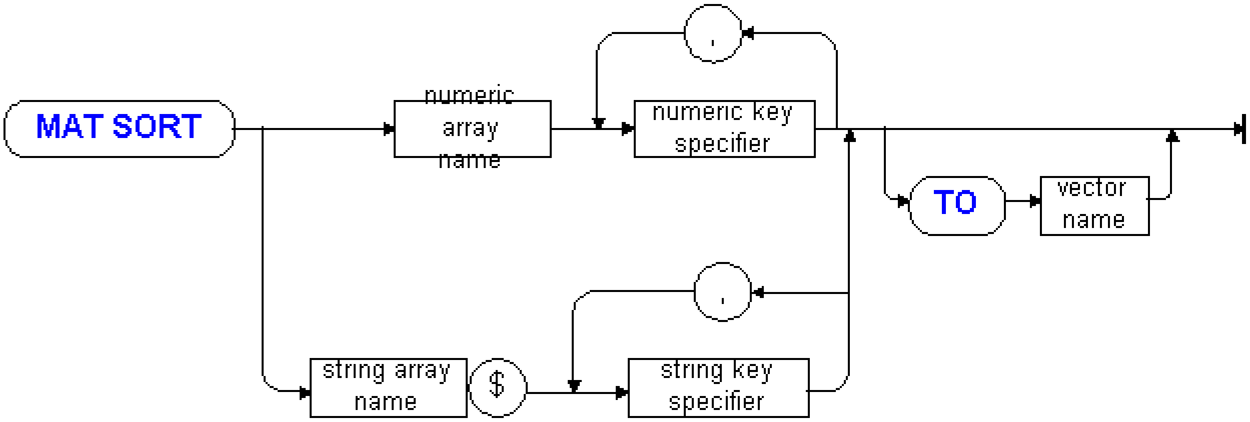
MAT, MAT REORDER, MAT SORT, REDIM

MAT SORT

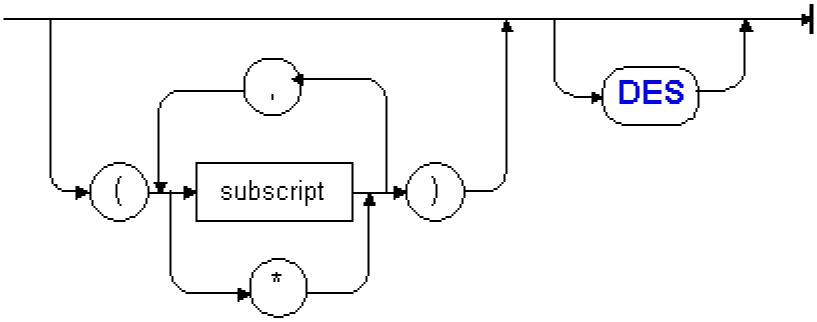
Sorts string or numeric array data.

Syntax: MAT SORT numeric-array numeric-keys [TO vector]
MAT SORT string-array$ string-keys [TO vector]

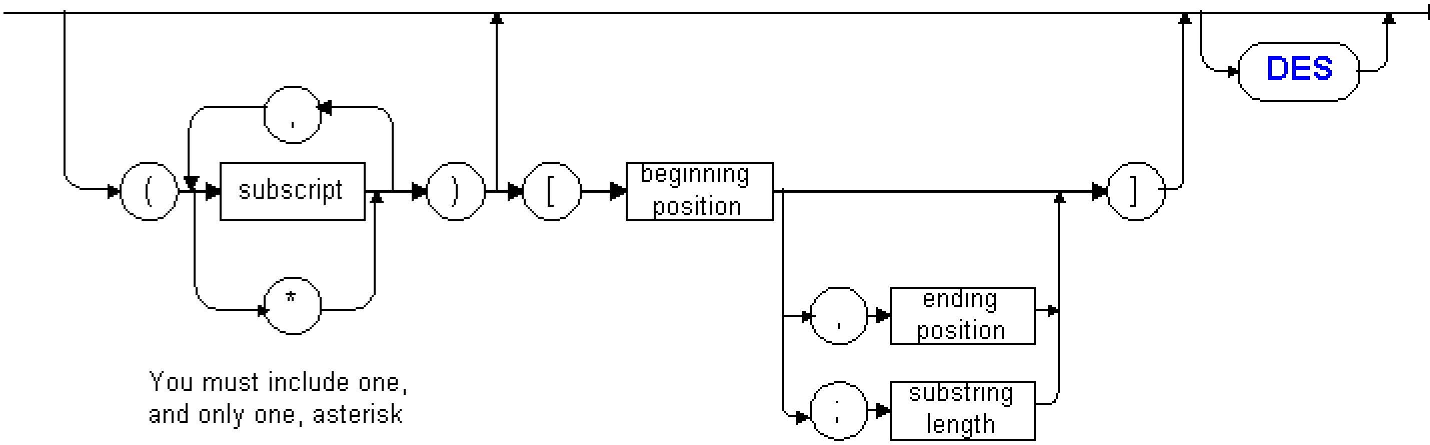
where: numeric-keys = (key-subscripts) [DES] [,numeric-keys]
key-subscripts = {subscript | *} [,key-subscripts]
The '*' must appear only once.
string-keys = string-key [,string-keys]
string-key = (key-subscripts) [sub-string] [DES]



numeric key specifier:



You must include one, and only one, asterisk string key specifier:



Sample: `MAT SORT A$(*)`

```
MAT SORT Array(Tag,*)
MAT SORT Vals(1,*,3),(2,*,5) DES
MAT SORT String$(*,2)[1;3] TO Order
```

View Sample: MAT SORT.BAS **(also found in examples directory)**

Description:

MAT SORT sorts a numeric or string array along one dimension. The direction of the sort is in ascending order unless the **DES** keyword follows the key specifier. For multi-dimensioned arrays, entire rows, columns, etc. are swapped in the ordering process according to the values in the sort key specifier.

The sort key specifier is made up of subscript values and an asterisk "*". The asterisk specifies the dimension to be sorted. The subscript values specify which array elements in that subscript are to be used during the sort. Sub-strings may be specified for string arrays.

The optional "**TO** vector" syntax stores the new order in a vector, leaving the original array unchanged. The vector is redimensioned to the size of the array dimension sorted. It is compatible with the MAT REORDER statement. It is best if the vector is an INTEGER array.

COMPLEX Arrays

MAT SORT can not sort a complex array since the concept of linear ordering does not apply to the complex plane. A complex array can be sorted indirectly by creating a REORDER vector that sorts the complex array according to some linear property of complex numbers, such as magnitude. In the following example, lines 90 to 110 sort the complex array C(*) according to magnitude. A similar technique can be used for other sorting criteria.

```
 10 COMPLEX C(1:8)
 20 REAL Abs(1:8)
 30 INTEGER I,Order(1:8)
 40 FOR I=1 TO 8 !Create array to sort
 50   C(I)=CMPLX(INT(RND*10),INT(RND*10))
 60 NEXT I
 70 PRINT USING "2(K,2X),/";C(*)
 80 ! Now sort by magnitude
 90 MAT Abs=ABS(C)
100 MAT SORT Abs(*) TO Order
110 MAT REORDER C BY Order
120 ! Print the result
130 FOR I=1 TO 8
140   PRINT C(I),ABS(C(I))
150 NEXT I
160 END
```

See Also:

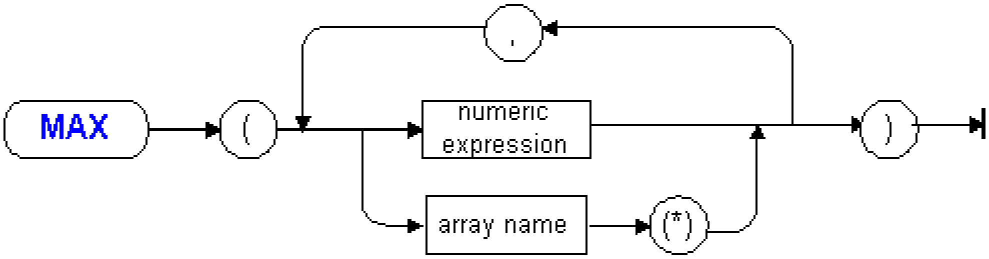
MAT, MAT REORDER, MAT SEARCH, REDIM

MAX

Returns the maximum value of a list of expressions.

Syntax: MAX(item [,item...])

where: item = numeric-expression | numeric-array(*)



Sample:

```
I=MAX(4,X,Y)
Largest=MAX(numerals(*))
PRINT MAX(First,20,Last/3)
Cost=MAX(Win1,Win2,Lose1)
```

View Sample: **MAX.BAS** **(also found in examples directory)**

Description:

The **MAX** numeric function returns the largest value of all the values in the argument list. If an item is an array it is treated as if each element in the array were an item.

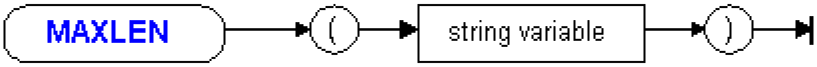
See Also:

MIN

MAXLEN

Gets maximum declared length of a string variable.

Syntax: MAXLEN(string-name$ [(*)|(subscripts)])



Sample: `MAXLEN(Newstring$)`

`Rows=MAXLEN(Alpharray$(*))`

View Sample: **MAXLEN.BAS** **(also found in examples directory)**

Description:

MAXLEN returns the declared length of the string variable as declared in an ALLOCATE, COM or DIM statement or an implicitly declared string variable.

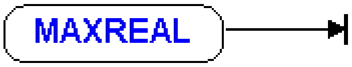
See Also:

BASE, DIM, RANK, SIZE

MAXREAL

Returns the largest positive REAL number.

Syntax: MAXREAL



Sample:
```
IF X>MAXREAL/Y THEN GOTO Overflow
```

View Sample: **MAXREAL.BAS** **(also found in examples directory)**

Description:

MAXREAL returns the largest positive REAL number that the computer can represent in its floating point number system. On computer systems that use the IEEE floating point number standard, the largest positive REAL number is approximately 1.797 693 134 862 32E+308.

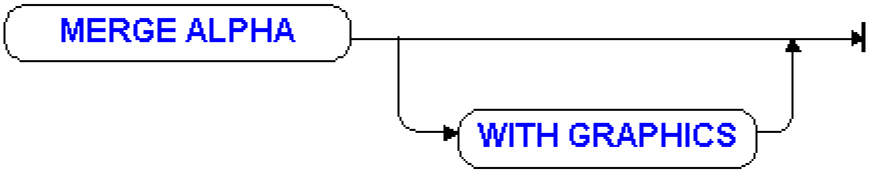
See Also:

MINREAL

MERGE ALPHA WITH GRAPHICS

Enables all planes for Alpha and Graphics.

Syntax: MERGE ALPHA [WITH GRAPHICS]



Sample: `IF Conf=4 THEN MERGE ALPHA WITH GRAPHICS`

View Sample: **MERGE ALPHA WITH GRAPHICS.BAS** **(also found in examples directory)**

Description:

This statement is the opposite of SEPARATE ALPHA FROM GRAPHICS. When merged, all bit-planes are used by both alpha and graphics. This means that alpha text is converted to graphic pixels and written into the graphic planes, overwriting any graphics data that might be present. Also, scrolling alpha text will scroll graphics, dumping either will dump both and the full range of colors are available for both alpha text and graphic output. **MERGE ALPHA** is the default mode.

See Also:

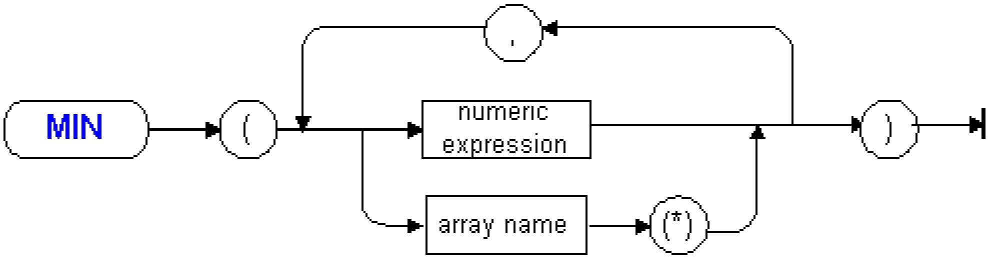
ALPHA, GRAPHICS, PLOTTER IS, SEPARATE ALPHA

MIN

Returns the minimum value of a list of expressions.

Syntax: MIN(item)

where: item = numeric-expression | numeric-array(*)



Sample:

```
I=MIN(4,3)
Small=MIN(Numerals(*))
PRINT MIN(First,20,Last/3)
```

View Sample: **MIN.BAS** **(also found in examples directory)**

Description:

The **MIN** numeric function returns the smallest value of all the items in the argument list. An array is treated as if all its elements were listed as items.

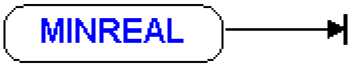
See Also:

MAX

MINREAL

Returns the smallest positive REAL number.

Syntax: MINREAL



Sample: `IF X<MINREAL*Y THEN GOTO Underflow`

View Sample: **MINREAL.BAS** **(also found in examples directory)**

Description:

MINREAL returns the smallest positive REAL number that the computer can represent in its floating point number system. On computer systems that use the IEEE floating point number standard, the smallest positive REAL number is approximately 2.225 073 858 507 24E-308.

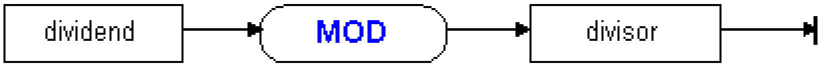
See Also:

MAXREAL

MOD

Returns remainder after integer division.

Syntax: dividend MOD divisor



Sample:

```
I=D MOD 16
PRINT "Inches"=";Length MOD 12
```

View Sample: **MOD.BAS** **(also found in examples directory)**

Description:

X **MOD** Y is the remainder from a division which produces an integral quotient and is defined as X - Y * (X DIV Y). If one or both of the operands are REAL, the result is REAL; otherwise the result is INTEGER. The difference between **MOD** and MODULO is explained in MODULO.

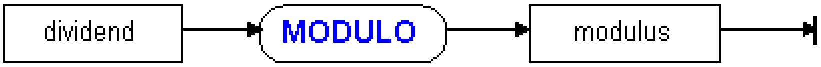
See Also:

DIV, INT, MODULO

MODULO

Returns the true mathematical modulus.

Syntax: dividend MODULO modulus



Sample:

```
I=D MODULO 16
PRINT "Inches" =";Length MODULO 12
R=12 MODULO −5
```

View Sample: MODULO.BAS **(also found in examples directory)**

Description:

X **MODULO** Y is defined as X - Y * INT(X/Y), where INT(X/Y) is the greatest integer less than or equal to X/Y. **MODULO** and MOD give the same result if both X and Y have the same sign, but differ if X and Y do not have the same sign. It can be seen why this is so from the definitions. (X DIV Y) divides and then converts to integer by truncation toward zero. INT(X/Y) divides and then converts to integer by truncation toward negative infinity.

See Also:

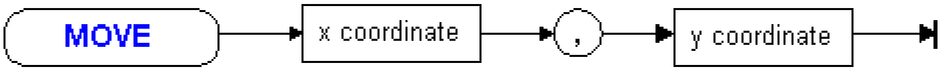
INT, MOD

MOVE

Moves the logical and physical pens to a new position.

Syntax: MOVE x-position, y-position

where: x-position, y-position = numeric-expressions



Sample: `MOVE 25,80`

`MOVE Newx,Newy`

View Sample: **MOVE.BAS** **(also found in examples directory)**

Description:

The pen is raised before being moved to the specified position. If both the current logical position and the specified position are outside the clip area the logical position is updated but no physical pen movement is made.

The PIVOT statement affects the **MOVE** statement.

See Also:

CLIP, DRAW, IDRAW, IMOVE, IPLOT, LINE TYPE, PIVOT, PLOT, RPLOT, SHOW, VIEWPORT, WINDOW

MOVELINES

Moves program lines from one location to another.

Syntax: MOVELINES start [,end] TO target

where: start, end and target = line-number | line-label

Sample:

```
MOVELINES 600 TO 1500
MOVELINES 500,1200 TO 4100
MOVELINES First,Second TO Target
```

Description:

MOVELINES moves a block of lines to a new location. This differs from the COPYLINES statement in that COPYLINES makes a copy of the original program portion. If no ending line is specified, only one line is moved. The target line cannot be in the range specified by start and end. If start doesn't exist, the line immediately after that line number is used. If end doesn't exist, the line immediately before that line number is used. If a non-existent line label is specified, an error will be reported. If the arguments specify a destination line number or program section that already exists, the old section will be renumbered to make room for the new program lines.

Line numbers and labels are renumbered and updated if needed. **MOVELINES** may not move lines containing a SUB program or DEF FN definition unless the new line number is greater than any existing line number; otherwise an error is issued because SUB or DEF FN must follow all previous lines. If an error occurs during a **MOVELINES**, the copy is terminated and the program is left partially changed. This command can only be executed from the keyboard. It cannot be included in a program.

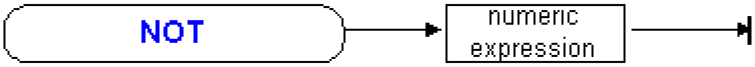
See Also:

CHANGE, COPYLINES, DEL, DELSUB, EDIT, FIND, INDENT, REN, SECURE, XREF

NOT

Returns the logical negation of an expression.

Syntax: NOT numeric-expression



Sample:

```
A=NOT 1
A=NOT B
IF NOT File_input THEN PRINT Prompt$
```

View Sample: **NOT.BAS** **(also found in examples directory)**

Description:

If the argument is zero, **NOT** returns a one. If the argument is non-zero, **NOT** returns a zero.

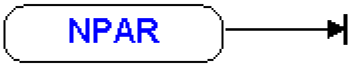
See Also:

AND, OR, EXOR

NPAR

Returns number of parameters passed to a subprogram.

Syntax: NPAR



Sample:

```
IF NPAR>5 THEN More
Global=NPAR-3
```

View Sample: **NPAR.BAS** **(also found in examples directory)**

Description:

NPAR is useful in subprograms with **OPTIONAL** parameters. **NPAR** can be used to determine which parameters were present in the calling argument list. An attempt to use a parameter which was not present results in an error. In the main program, **NPAR** returns a zero.

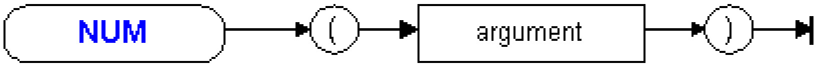
See Also:

CALL, DEF FN, FN, SUB

NUM

Returns the decimal ASCII equivalent of the first character in a string.

Syntax: NUM(string-expression)



Sample:

```
A=NUM(B$)
A=NUM("0")
N=NUM(Alph$)
B=NUM(B$[V])/16
```

View Sample: **NUM.BAS** **(also found in examples directory)**

Description:

The range of the returned values is 0 through 255.

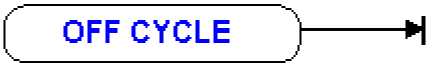
See Also:

CHR$, LWC$, REV$, RPT$, POS, TRIM$, UPC$, VAL, VAL$

OFF CYCLE

Cancels event branches defined by ON CYCLE.

Syntax: OFF CYCLE



Sample:

```
OFF CYCLE
IF Complete THEN OFF CYCLE
```

View Sample: **OFF CYCLE.BAS** **(also found in examples directory)**

Description:

Any CYCLE events that have been logged but not yet serviced, are canceled.

Execution of an **OFF CYCLE** statement within a subprogram will disable the ON CYCLE definition within the context of the subprogram, but when control is returned to the calling program the ON CYCLE definition is re-enabled.

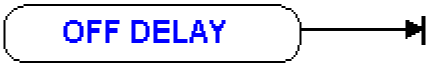
See Also:

ENABLE, DISABLE, ON CYCLE, SYSTEM PRIORITY

OFF DELAY

Cancels event branches defined by ON DELAY.

Syntax: OFF DELAY



Sample:

```
OFF DELAY
IF Finish THEN OFF DELAY
```

View Sample: **OFF DELAY.BAS** **(also found in examples directory)**

Description:

Any DELAY events that have been logged but not yet serviced, are canceled.

Execution of an **OFF DELAY** statement within a subprogram will disable the ON DELAY definition within the context of the subprogram, but when control is returned to the calling program the ON DELAY definition is re-enabled.

See Also:

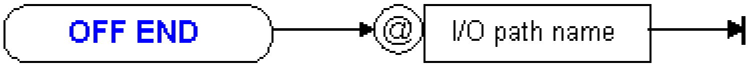
ENABLE, DISABLE, ON DELAY, SYSTEM PRIORITY

OFF END

Cancels event branches defined by ON END.

Syntax: OFF END @io-path

where: io-path = name assigned to a data file



Sample:

```
             OFF END @File
IF Finish THEN OFF END @Input
```

View Sample: **OFF END.BAS** **(also found in examples directory)**

Description:

Execution of an **OFF END** statement within a subprogram will disable the ON END definition within the context of the subprogram, but when control is returned to the calling program the ON END definition is re-enabled.

End-of-file and end-of-record errors will be reported if no ON END definition is active.

See Also:

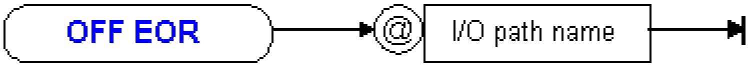
ENABLE, DISABLE, ON END, SYSTEM PRIORITY

OFF EOR

Cancels event branches defined by ON EOR.

Syntax: OFF EOR @non-buf-io-path

where: non-buf-io-path = io-path used in the ON EOR statement



Sample:

```
OFF EOR @Dev
IF Finish THEN OFF EOR @File
```

Description:

Any End-of-Record (EOR) events that have been logged but not yet serviced, are canceled. Executing **OFF EOR** within a subprogram disables the ON EOR definition within that subprogram context. When control is returned to the calling program, any pre-existent ON EOR definition is re-enabled.

See Also:

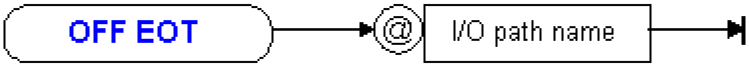
ABORTIO, ON EOR, ON EOT, TRANSFER, WAIT

OFF EOT

Cancels event branches defined by ON EOT.

Syntax: OFF EOT @non-buf-io-path

where: non-buf-io-path = io-path used in the ON EOT statement



Sample:

```
OFF EOT @Dev
IF Finis THEN OFF EOT @File
```

Description:

Any End-of-Transfer (EOT) events that have been logged but not yet serviced, are canceled. Executing **OFF EOT** within a subprogram disables the ON EOT definition within that subprogram context. When control is returned to the calling program, any pre-existent ON EOT definition is re-enabled.

See Also:

ABORTIO, ON EOR, ON EOT, TRANSFER, WAIT

OFF ERROR

Cancels event branches defined by ON ERROR.

Syntax: OFF ERROR

Sample: `IF Finish THEN OFF ERROR`

View Sample: **OFF ERROR.BAS** **(also found in examples directory)**

Description:

Execution of an **OFF ERROR** statement will cause any subsequent errors to be reported to the user and program execution will PAUSE.

See Also:

ENABLE, DISABLE, ON INTR, SYSTEM PRIORITY

OFF EVENT

Cancels event branches defined by ON EVENT.

Syntax: OFF EVENT

Sample:

```
OFF EVENT @Pushbutton_3,"ACTIVATED"
OFF EVENT @Slider, “DONE”
```

Description:

The OFF EVENT statement undefines and disables a widget event that was defined and enabled earlier by an ON EVENT statement. There are three important differences between the OFF EVENT and DISABLE EVENT statements:

- DISABLE EVENT temporarily disables the event, whereas OFF EVENT permanently deactivates the event.
- Only one occurrence of the event will be logged if the event is disabled with a DISABLE EVENT statement. Therefore, the branch will be taken once the event is re-enabled with an ENABLE EVENT statement.
- The event will NOT be logged and the branch will never be taken if the event is deactivated with an OFF EVENT statement.

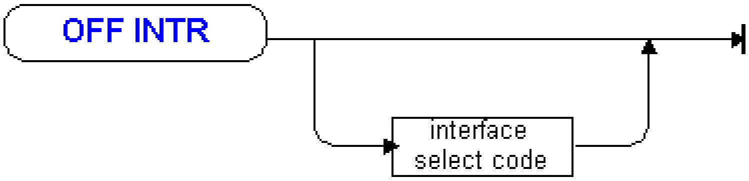
See Also:

ENABLE EVENT, DISABLE EVENT, ON EVENT

OFF INTR

Cancels event branches defined by ON INTR.

Syntax: OFF INTR [interface-select-code]



Sample:

```
OFF INTR
OFF INTR 10
OFF INTR Gpib
```

View Sample: **OFF INTR.BAS** **(also found in examples directory)**

Description:

Any INTR events that have been logged but not yet serviced are canceled.

An **OFF INTR** statement without the optional interface select code disables event-initiated branches on all devices. If the interface select code is specified, only that interface interrupt will be disabled.

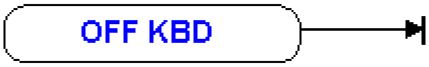
See Also:

ENABLE, ENABLE INTR, DISABLE, DISABLE INTR, ON-EVENT, SYSTEM PRIORITY

OFF KBD

Cancels event branches defined by ON KBD.

Syntax: OFF KBD



Sample: `IF Finish THEN OFF KBD`

View Sample: **OFF KBD.BAS** **(also found in examples directory)**

Description:

Any KBD events that have been logged but not yet serviced are canceled and the keyboard buffer is cleared.

Execution of an **OFF KBD** statement within a subprogram will disable the ON KBD definition within the context of the subprogram, but when control is returned to the calling program the ON KBD definition is re-enabled. The keyboard buffer remains cleared.

See Also:

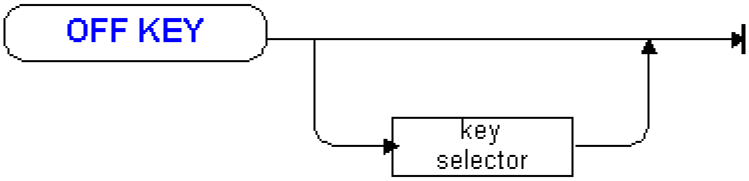
ENABLE, DISABLE, KBD$, ON KBD, SYSTEM PRIORITY

OFF KEY

Cancels event branches defined by ON KEY.

Syntax: OFF KEY [key-number]

where: key-number = numeric-expression rounded to an integer



Sample:

```
OFF KEY
OFF KEY 2
OFF KEY Lock
IF Carkey AND NOT Housekey THEN OFF KEY
```

View Sample: **OFF KEY.BAS** **(also found in examples directory)**

Description:

An **OFF KEY** statement without the key-number cancels event branches for all softkeys. If the key-number is specified then only that softkey will be canceled. The key-number range is zero through twenty-three. Any KEY events for affected softkeys that have been logged but not yet serviced are canceled. **OFF KEY** also restores the previous key labels.

Executing **OFF KEY** within a subprogram disables the ON KEY definitions within the subprogram context. When control is returned to the calling program the ON KEY definitions are re-enabled.

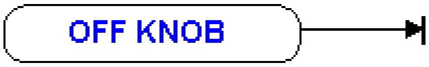
See Also:

ENABLE, DISABLE, ON KEY, SYSTEM PRIORITY

OFF KNOB

Cancels event branches defined by ON KNOB.

Syntax: OFF KNOB



Sample: `IF Scroll THEN OFF KNOB`

View Sample: **OFF KNOB.BAS** **(also found in examples directory)**

Description:

Any KNOB events that have been logged but not yet serviced are canceled. After **OFF KNOB**, the knob or mouse will scroll the screen and move the cursor.

See Also:

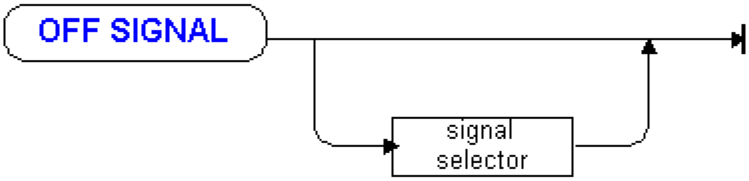
ENABLE, DISABLE, KNOBX, KNOBY, ON KNOB, SYSTEM PRIORITY

OFF SIGNAL

Cancels event branches defined by ON SIGNAL.

Syntax: OFF SIGNAL [signal-number]

where: signal-number = numeric-expression rounded to an integer



Sample:

```
OFF SIGNAL
OFF SIGNAL 5
OFF SIGNAL Msg
```

View Sample: **OFF SIGNAL.BAS** **(also found in examples directory)**

Description:

An **OFF SIGNAL** statement without the signal number will cancel all the ON SIGNAL definitions. If the signal number is specified then only that signal will be canceled. The signal-number has a range of zero through fifteen. Any SIGNAL events with the same signal number that have been logged but not yet serviced are canceled. **OFF SIGNAL** applies to the current context only.

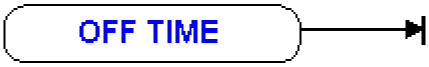
See Also:

ENABLE, DISABLE, ON SIGNAL, SIGNAL, SYSTEM PRIORITY

OFF TIME

Cancels event branches defined by ON TIME.

Syntax: OFF TIME



Sample: `IF Clock THEN OFF TIME`

View Sample: **OFF TIME.BAS** **(also found in examples directory)**

Description:

Any TIME events that have been logged but not yet serviced are canceled.

Execution of an **OFF TIME** statement within a subprogram will cancel the ON TIME definition within the context of the subprogram, but when control is returned to the calling program the ON TIME definition is re-enabled.

See Also:

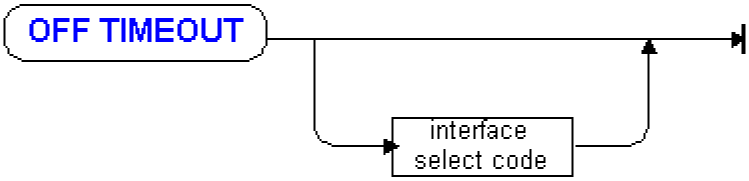
ENABLE, DISABLE, ON TIME, SYSTEM PRIORITY

OFF TIMEOUT

Cancels event branches defined by ON TIMEOUT.

Syntax: OFF TIMEOUT [interface-select-code]

where: interface-select-code = integer numeric-expression



Sample: `OFF TIMEOUT`

```
OFF TIMEOUT 8
OFF TIMEOUT Gpib
```

View Sample: **OFF TIMEOUT.BAS** **(also found in examples directory)**

Description:

No more timeouts can occur on the affected interfaces after an **OFF TIMEOUT** statement.

An **OFF TIMEOUT** statement without the interface-select-code will cancel the ON TIMEOUT definitions on all interfaces. If the interface-select-code is specified then only that interface **TIMEOUT** will be canceled.

See Also:

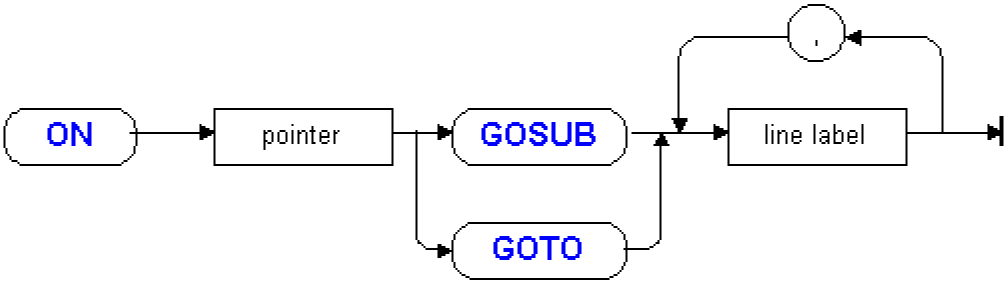
ENABLE, DISABLE, ON TIMEOUT, SYSTEM PRIORITY

ON

Transfers control to one of a list of lines.

Syntax: ON index {GOSUB | GOTO} line [,line...]

where: index = numeric-expression rounded to an integer
line = line-number | line-label



Sample:
```
ON Choose GOSUB Placea,Placeb
ON X/2 GOTO 700,800,900
```

View Sample: **ON.BAS** **(also found in examples directory)**

Description:

ON ... GOTO or **ON ... GOSUB** allows you to perform a multi-way transfer. You can select one of a list of program line numbers by the computed value of a numeric expression. The numeric expression is rounded to an integer value and is used as an index to select one of the line numbers from the list.

If the integer value is 1, the first line number is used. If the integer value is 2, the second line number is used and so on. If the index number is less than one or greater than the number of line numbers in the list, an error is generated.

If GOSUB is specified the matching RETURN is to the line following the **ON** statement.

See Also:

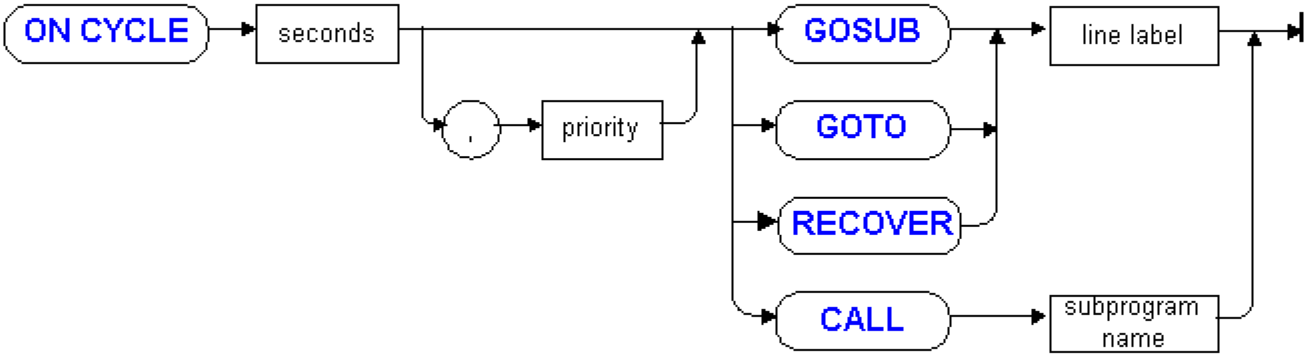
GOTO, GOSUB, RETURN

ON CYCLE

Defines a repeating event branch.

Syntax: ON CYCLE seconds [,priority] action

where: seconds = numeric-expression rounded to an integer.
action = { GOTO|GOSUB|RECOVER } line | CALL subprogram
line = line-number | line-label



Sample:

```
ON CYCLE Seconds,Priority CALL Sub
ON CYCLE Max RECOVER Names
ON CYCLE 1200,3 GOTO 2000
```

View Sample: **ON CYCLE.BAS** **(also found in examples directory)**

Description:

ON CYCLE defines a repeating event branch. After the specified number of seconds has passed, an event is generated and the cycle is started again. The value of seconds can range from 0.01 to 167772.16 but is rounded to the timing resolution of the computer. If short **CYCLE** values cause events to occur faster than the computer can service them, some events will be lost.

There is only one **CYCLE** timer. Executing a new **ON CYCLE** while another **ON CYCLE** is still in effect will cause the **CYCLE** timer to use the new seconds value. If the **ON CYCLE** is executed in a different program context the original **ON CYCLE** definition is restored when control returns to the calling context. The old **CYCLE** time is not restored, however.

ON CYCLE is canceled by OFF CYCLE and disabled by DISABLE. A SUBEXIT, SUBEND, or RETURN from the defining subprogram also cancels it.

Common Information

The following information is common to the **ON CYCLE**, DELAY, EOR, EOT, INTR, KBD, KEY, KNOB, SIGNAL, TIME statements.

The line number or line label following the GOTO, GOSUB or **RECOVER** or the subprogram name following the CALL indicates where to transfer control when the event occurs. Line numbers or labels must be in the same subprogram as the **ON** statement. When returning from a CALL or GOSUB execution continues with the line that would have executed next when the event occurred. **RECOVER** causes the program to SUBEXIT from subprograms as needed to return to the defining subprogram and then does a GOTO to the specified program line. (The defining subprogram is the subprogram with the **ON** statement.)

The event branch can only occur if the current SYSTEM PRIORITY is less than the priority specified in the **ON** statement. The default priority is one. The highest priority that can be specified is fifteen. ON END, ON ERROR and ON TIMEOUT events have a higher priority than all other events. If an event branch can not take place because of system priority, the event is logged and occurs later when the system priority drops to a level which allows it.

When an event branch is taken the system priority is changed depending on the branch type. With a GOTO the system priority is not changed. With a **RECOVER** the system priority is only changed if any SUBEXITs are performed, in which case the system priority is restored to the value when the defining subprogram called another subprogram. With a CALL or GOSUB the system priority is changed to the specified priority. When returning from the CALL or GOSUB the system priority is restored to the value before the branch was taken.

If other subprograms have been called from the defining subprogram when the event occurs, when the branch can be taken depends on the branch type. CALL or **RECOVER** branches can still occur as soon as the event occurs. (Although branches are not taken in the middle of execution of a line; the branch is taken between lines.) GOTO or GOSUB branches can not be taken immediately. The event will be logged and then serviced when control returns to the defining subprogram.

See Also:

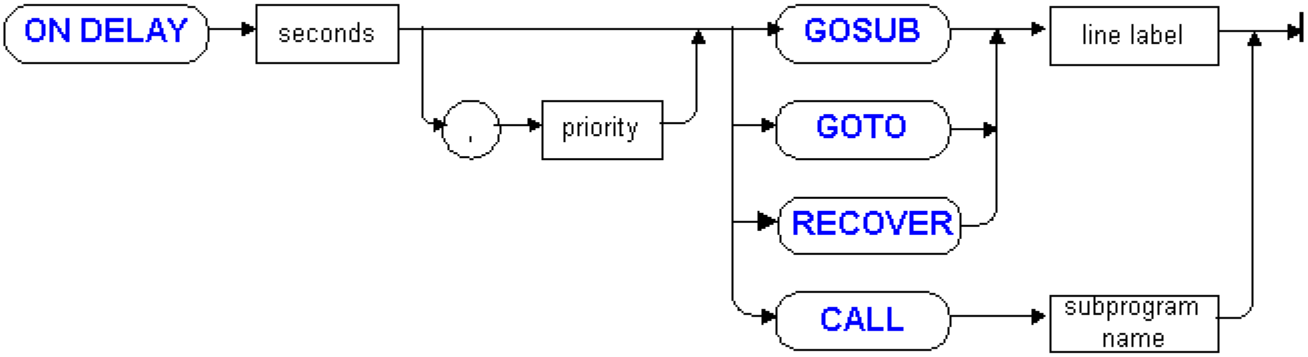
ENABLE, DISABLE, OFF CYCLE, SYSTEM PRIORITY

ON DELAY

Defines an event branch after specified seconds.

Syntax: ON DELAY seconds [,priority] action

where: seconds = numeric-expression rounded to an integer.
action = { GOTO|GOSUB|RECOVER } line | CALL subprogram
line = line-number | line-label



Sample:

```
ON DELAY Seconds,Priority CALL Sub1
ON DELAY 3 GOTO 5710
ON DELAY Maxtime,4 GOSUB Branch
```

View Sample: ON DELAY.BAS **(also found in examples directory)**

Description:

ON DELAY defines a one time event branch to take after a specified number of seconds. The value of seconds can range from 0.01 to 167772.16 but is rounded to the timing resolution of the computer.

There is only one **DELAY** timer. Executing a new **ON DELAY** while another **ON DELAY** is still in effect will cause the **DELAY** timer to use the new seconds value. If the **ON DELAY** is executed in a different program context, the original **ON DELAY** definition is restored when control returns to the calling context. The old **DELAY** time is not restored, however.

ON DELAY is canceled by OFF DELAY and disabled by DISABLE. A SUBEXIT, SUBEND, or RETURN from the defining subprogram also cancels it.

More information about **ON DELAY** can be found under the "Common Information" heading of the ON CYCLE manual entry.

See Also:

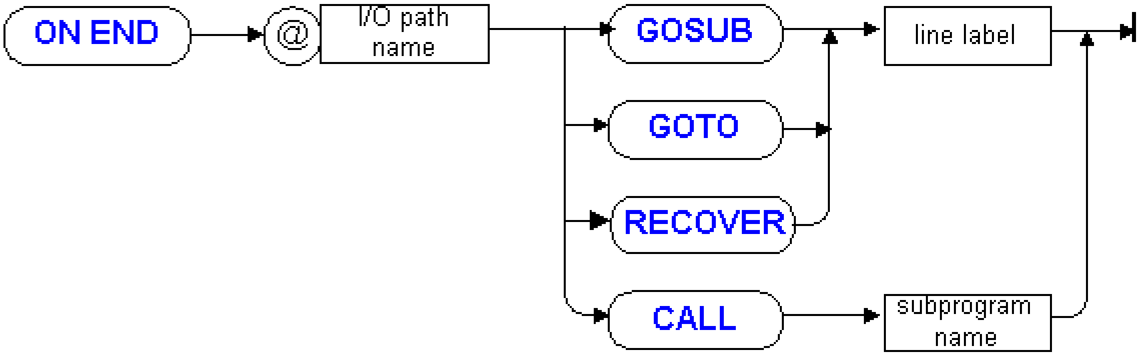
ENABLE, DISABLE, OFF DELAY, SYSTEM PRIORITY

ON END

Defines an event branch for end-of-file conditions.

Syntax: ON END @io-path action

where: action = { GOTO|GOSUB|RECOVER } line | CALL subprogram
line = line-number | line-label



Sample:

```
ON END @Dat GOTO 750
ON END @Code CALL Find
ON END @File RECOVER Fix
```

View Sample: **ON END.BAS** **(also found in examples directory)**

Description:

When you ENTER data and there is no more data in a file, or when a random access OUTPUT or ENTER requires more bytes than the record size, an end-of-file error occurs which may be caught by the **ON END** statement. The **ON END** statement must be executed before the end-of-file error condition occurs. If an **ON END** event handler does not exist, error 59 occurs, which can be trapped like other errors with an ON ERROR handler.

ON END is canceled by OFF END but is not disabled by DISABLE. A SUBEXIT, SUBEND, or RETURN from the defining subprogram also cancels it.

When returning from a CALL or GOSUB execution continues with the line following the line causing the end-of-file.

Common Information for ON END, ERROR, TIMEOUT

The line number or line label following the GOTO, GOSUB, or **RECOVER** or the subprogram name following the CALL indicates where to transfer control when the event occurs. Line numbers or labels must be in the same subprogram as the **ON** statement. **RECOVER** causes the program to SUBEXIT from subprograms as needed to return to the defining subprogram and then does a GOTO to the specified program line. (The defining subprogram is the subprogram with the **ON** statement.)

The **ON END** and ON TIMEOUT events have a fixed priority of fifteen and ON ERROR has a fixed priority of seventeen. However, when one of these events occurs, the current SYSTEM PRIORITY is ignored and the branch occurs immediately. The only exception is when an error occurs when the system priority is already seventeen; this "double fault" condition can not be trapped.

When an event branch is taken the system priority is changed depending on the branch type. With a GOTO the system priority is not changed. With a **RECOVER** the system priority is only changed if any SUBEXITs are performed, in which case the system priority is restored to the value when the defining subprogram called another subprogram. With a CALL or GOSUB the system priority is changed to fifteen for **ON END** and ON TIMEOUT or seventeen for ON ERROR. When returning from the CALL or GOSUB the system priority is restored to the value before the branch was taken.

If other subprograms have been called from the defining subprogram when the event occurs, the action taken depends on the branch type. CALL or **RECOVER** branches can still occur as soon as the event occurs. (Although branches are not taken in the middle of execution of a line; the branch is taken between lines.) GOTO or GOSUB branches can not be taken so an error occurs.

See Also:

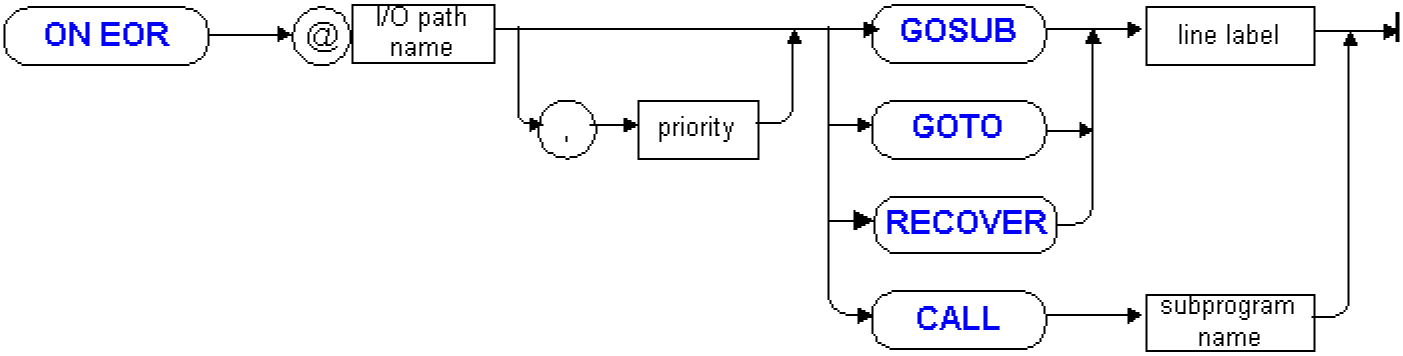
ERRL, ERRLN, ERRM$, ERRN, ON ERROR, ON TIMEOUT, OFF END

ON EOR

Defines an event branch for end-of-record conditions.

Syntax: ON EOR @io-path [,priority] action

where: action = { GOTO|GOSUB|RECOVER } line | CALL subprogram
line = line-number | line-label



Sample:

```
ON EOR @Dev GOTO 1200
ON EOR @Code,2 CALL Record
```

Description:

The TRANSFER statement can define what is to be considered a record for the purpose of that particular TRANSFER. When an end-of-record is detected, an **EOR** event occurs which may be caught by the **ON EOR** statement. The **ON EOR** statement must be executed before the end-of-record condition occurs.

The I/O path must be the I/O path used in the TRANSFER to specify the device. Using the I/O path assigned to the buffer will cause an error.

If another **ON EOR** is executed in a different program context, the original **ON EOR** definition is restored when control returns to the calling context.

ON EOR is canceled by OFF EOR and is disabled by DISABLE. A SUBEXIT, SUBEND, or RETURN from the defining subprogram also cancels it. If a context exit is delayed until a TRANSFER terminates, any **EOR** events generated during the delay are discarded. Use WAIT FOR EOR to force the event to be serviced before the subprogram exits.

More information about **ON EOR** can be found under the "Common Information" heading of the ON CYCLE manual entry.

See Also:

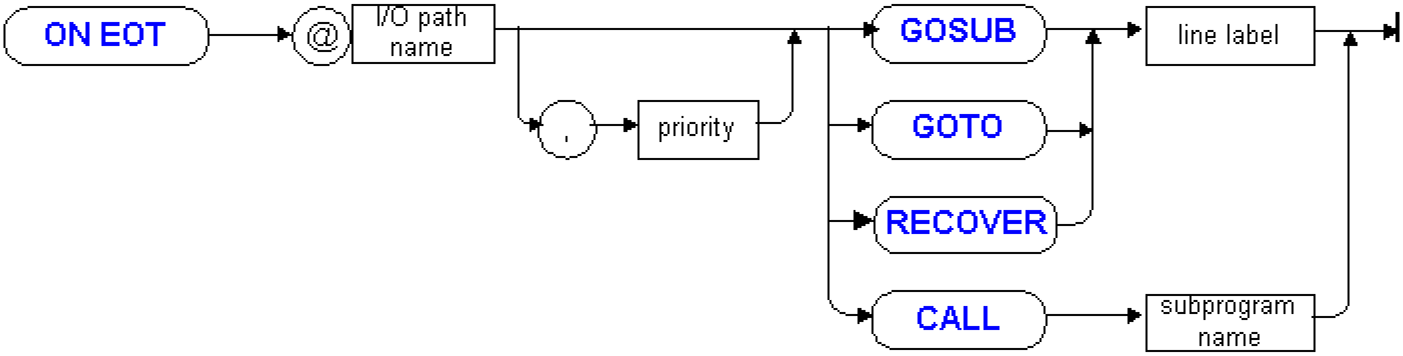
ABORTIO, OFF EOR, ON EOT, TRANSFER, WAIT

ON EOT

Defines an event branch for end-of-transfer conditions.

Syntax: ON EOT @io-path [,priority] action

where: action = { GOTO|GOSUB|RECOVER } line | CALL subprogram
line = line-number | line-label



Sample:

```
ON EOT @Dev GOTO 1200
ON EOT @Code,2 CALL Done
```

View Sample: **ON EOT.BAS** **(also found in examples directory)**

Description:

When a TRANSFER finishes, an end-of-transfer, **EOT**, event occurs which may be caught by the **ON EOT** statement. The **ON EOT** statement must be executed before the TRANSFER ends.

The I/O path must be the I/O path used in the TRANSFER to specify the device. Using the I/O path assigned to the buffer will cause an error.

If another **ON EOT** is executed in a different program context, the original **ON EOT** definition is restored when control returns to the calling context.

ON EOT is canceled by OFF EOT and is disabled by DISABLE. A SUBEXIT, SUBEND, or RETURN from the defining subprogram also cancels it. If a context exit is delayed until a TRANSFER terminates, any **EOT** events generated during the delay are discarded. Use WAIT FOR EOT to force the event to be serviced before the subprogram exits.

More information about **ON EOT** can be found under the "Common Information" heading of the ON CYCLE manual entry.

See Also:

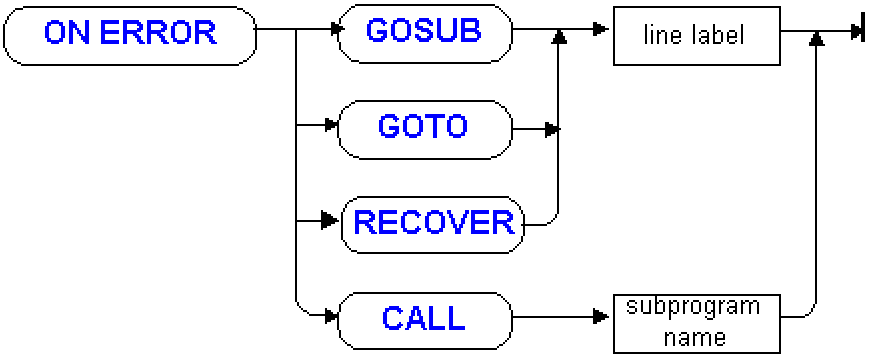
ABORTIO, OFF EOT, ON EOR, TRANSFER, WAIT

ON ERROR

Defines an event branch for trappable errors.

Syntax: ON ERROR action

where: action = { GOTO|GOSUB|RECOVER } line | CALL subprogram
line = line-number | line-label



Sample:

```
ON ERROR GOTO 2000
ON ERROR CALL Ertrap
ON ERROR RECOVER Test
```

View Sample: **ON ERROR.BAS (also found in examples directory)**

Description:

The **ON ERROR** statement specifies an error handling routine to be called when an error occurs during program execution. The **ON ERROR** statement must be executed before the error condition occurs. The routine can evaluate the error condition by using the ERRL, ERRLN and ERRN, functions and any other pertinent information to determine the corrective action to take. If there is not enough memory to run the routine, the original error is reported to the user and the program is paused.

If another **ON ERROR** is executed in a different context, the original **ON ERROR** definition is restored when control returns to the calling context. **ON ERROR** is canceled by OFF ERROR but is not disabled by DISABLE. A SUBEXIT, SUBEND or RETURN from the defining subprogram also cancels it.

When returning from a CALL or GOSUB execution normally continues with the offending line. If the error handling routine does not correct the cause of the error, the error will occur again, causing an infinite loop. To avoid re-execution of the line, use ERROR SUBEXIT instead of SUBEXIT or ERROR RETURN instead of RETURN.

If an error occurs in an error handling routine called with GOSUB or CALL, it is reported to the user and the program is paused. If an error occurs in an error handling routine called with GOTO or **RECOVER**, an infinite loop can result.

If **ON ERROR** is not used to handle an error, the program is paused and an error message is displayed on the message line. Pressing CONTINUE will re-execute the offending line. Type CONT followed by the line number of the next line to continue execution without re-executing the offending line.

More information about **ON ERROR** can be found under the "Common Information" heading of the ON END manual entry.

See Also:

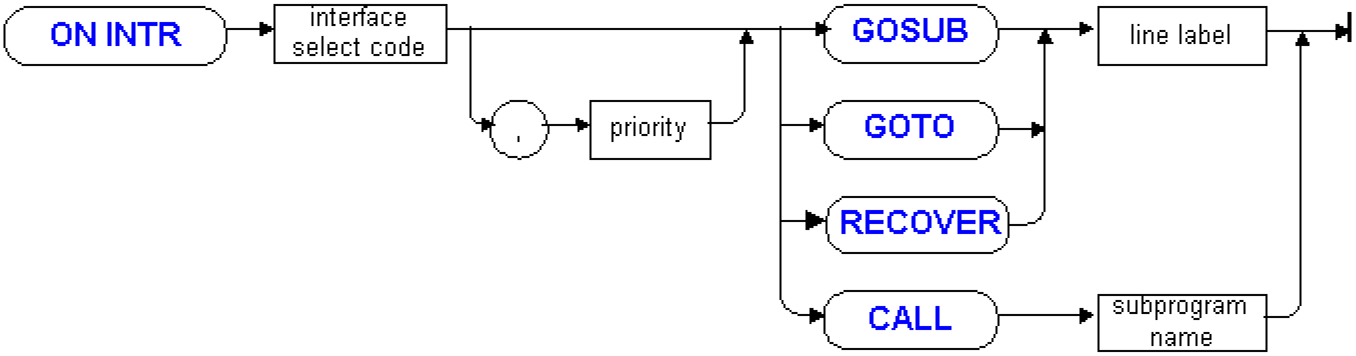
CAUSE ERROR, CLEAR ERROR, ERRL, ERRLN, ERRM$, ERRN, ERROR RETURN, ERROR SUBEXIT, ON END, ON TIMEOUT

ON INTR

Defines a hardware interrupt initiated branch.

Syntax: ON INTR interface-select-code [,priority] action

where: action = { GOTO|GOSUB|RECOVER } line | CALL subprogram
line = line-number | line-label



Sample:

```
ON INTR 7 GOTO 1000
ON INTR Isc,Priority CALL Sub
ON INTR Gpib,4 GOSUB Repair
```

View Sample: **ON INTR.BAS** **(also found in examples directory)**

Description:

ON INTR defines an event branch to be taken when an interface card generates an interrupt. Execution of an **ON INTR** statement is not sufficient to allow an interrupt to occur. As a minimum, ENABLE INTR must be executed to establish an interrupt mask. Depending on the interface, additional statements may have to be executed as well. Refer to the device driver documentation for more information.

When an interrupt occurs a DISABLE INTR for the interface is automatically executed. Consequently, an ENABLE INTR statement must be used to explicitly re-enable interrupts.

There is only one ENABLE INTR mask per interface select code. Executing a new ENABLE INTR while another is still in effect will cause the interface or device to use the new mask value. If the **ON INTR** is executed in a different program context, the original **ON INTR** definition is restored when control returns to the calling context. The ENABLE INTR mask is not restored, however.

ON INTR is canceled by OFF INTR and disabled by DISABLE or DISABLE INTR. A SUBEXIT, SUBEND or RETURN from the defining subprogram also cancels it.

More information about **ON INTR** can be found under the "Common Information" heading of the ON CYCLE manual entry.

See Also:

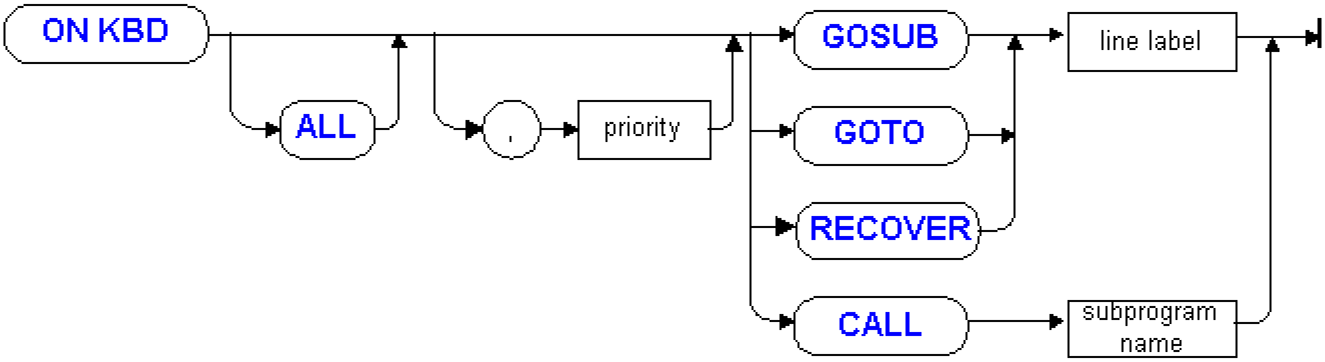
ENABLE, ENABLE INTR, DISABLE, DISABLE INTR, OFF INTR, SYSTEM PRIORITY

ON KBD

Defines an event branch for when a key is pressed.

Syntax: ON KBD [ALL] [,priority] action

where: action = { GOTO|GOSUB|RECOVER } line | CALL subprogram
line = line-number | line-label



Sample:

```
ON KBD GOTO 2000
ON KBD,Order GOSUB First
ON KBD ALL RECOVER 500
ON KBD ALL,3 CALL Sub
```

View Sample: **ON KBD.BAS** **(also found in examples directory)**

Description:

ON KBD defines an event branch to be taken when a key is pressed. **ON KBD ALL** traps all alpha-numeric keys and HTBasic function keys except RESET. The following keys are not trapped if **ALL** is not specified: CLR I/O, MENU, PAUSE, s-MENU, STOP, EXECUTE, USER and any softkeys.

If **ON KBD** is active, immediate execution of keyboard editing and display control function keys is suspended. All keystrokes go into a special KBD$ buffer. The buffer is cleared when it is read. The event handling routine can selectively execute keys found in KBD$ by including them in an OUTPUT KBD statement:

```
OUTPUT KBD;Buf$;
```

Unless an ON KNOB definition is active, movement of the mouse generates **ON KBD** interrupts and places UP, DOWN, LEFT or RIGHT keystrokes into the KBD$ buffer. If both **ON KBD ALL** and ON KEY are active, **ON KBD ALL** takes precedence over ON KEY.

Executing a new **ON KBD** while another **ON KBD** is still in effect overrides the previous **ON KBD** definition. If the **ON KBD** is executed in a different program context, the original **ON KBD** definition is restored when control returns to the calling context.

ON KBD is canceled by OFF KBD, disabled by DISABLE and temporarily disabled by an LINPUT, INPUT, or ENTER KBD statement. A SUBEXIT, SUBEND, or RETURN from the defining subprogram also cancels it.

More information about **ON KBD** can be found under the "Common Information" heading of the ON CYCLE manual entry.

See Also:

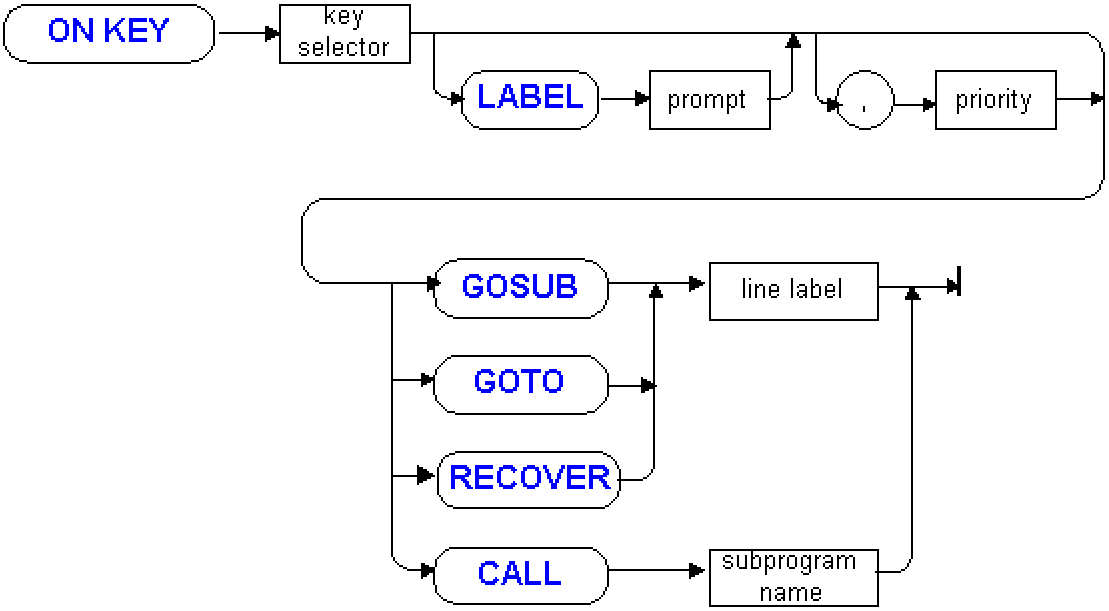
ENABLE, DISABLE, KBD$, OFF KBD, SYSTEM PRIORITY

ON KEY

Defines an event branch for when a softkey is pressed.

Syntax: ON KEY key-number [LABEL label] [,priority] action

where: key-number = numeric-expression rounded to an integer.
label = string-expression
action = { GOTO|GOSUB|RECOVER } line | CALL subprogram
line = line-number | line-label



Sample:

```
ON KEY 1 GOTO 200
ON KEY 5 LABEL Find$ RECOVER 500
ON KEY 2 LABEL "Print",3 CALL Findings
```

View Sample: **ON KEY.BAS** **(also found in examples directory)**

Description:

ON KEY defines a softkey event branch and optionally a label to be displayed in the softkey menu. When the softkey is pressed, the event occurs. The key number must be in the range of zero through twenty-three. Only as many characters as will fit in the menu area softkey label are displayed from the label.

If the label begins with a CLR LN key (CHR$(255) & "#"), only the characters after the CLR LN will be displayed. If the label begins with a CONTINUE key, the two characters (CHR$(255) & "C") will be replaced with the string "CONTINUE". If the label begins with a RUN key, the two characters (CHR$(255) & "R") will be replaced with the string "RUN".

Executing a new **ON KEY** while another **ON KEY** for the same softkey is still in effect will override the previous LABEL and definition. If the **ON KEY** is executed in a different program context, the original **ON KEY** definition is restored when control returns to the calling context.

ON KEY is canceled by OFF KEY, disabled by DISABLE and temporarily disabled by an LINPUT, INPUT, or ENTERKBD statement. A SUBEXIT, SUBEND, or RETURN from the defining subprogram also cancels it.

More information about **ON KEY** can be found under the "Common Information" heading of the ON CYCLE manual entry.

See Also:

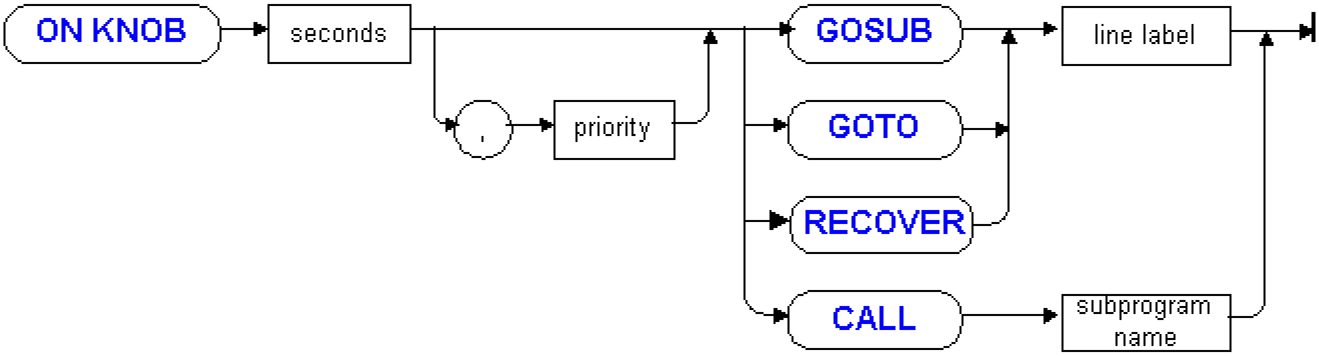
ENABLE, DISABLE, OFF KEY, SET KEY, SYSTEM PRIORITY

ON KNOB

Defines an event branch for when the KNOB is turned.

Syntax: ON KNOB seconds [,priority] action

where: action = { GOTO|GOSUB|RECOVER } line | CALL subprogram
line = line-number | line-label



Sample:

```
ON KNOB 1 GOTO 500
ON KNOB Seconds,Priority Call Sub
ON KNOB 1/2,4 GOSUB Label
```

View Sample: ON KNOB.BAS **(also found in examples directory)**

Description:

ON KNOB specifies the time interval in seconds for which movement of the **KNOB** is sampled. Nothing happens, however, until the first time the **KNOB** is moved after the **ON KNOB** statement has been executed. Once initial movement of the **KNOB** is detected, a timer begins for the specified interval. When the interval has expired, KNOBX and KNOBY are set to the distance the **KNOB** moved during the interval. A **KNOB** event is then generated. The value of seconds can range from 0.01 to 2.55 but is rounded to the timing resolution of the computer.

The KNOBX and KNOBY functions are read to determine the number of increments the **KNOB** has been moved in the x and the y directions during the interval.

Executing a new **ON KNOB** while another **ON KNOB** is still in effect overrides the previous **ON KNOB**definition. If the **ON KNOB** is executed in a different program context, the original **ON KNOB** definition is restored when control returns to the calling context.

ON KNOB is canceled by OFF KNOB and disabled by DISABLE. A SUBEXIT, SUBEND, or RETURN from the defining subprogram also cancels it.

While the syntax of this statement specifies a knob, typically a mouse is used instead; the syntax is for compatibility with older versions of HP BASIC.

More information about **ON KNOB** can be found under the "Common Information" heading of the ON CYCLE manual entry.

See Also:

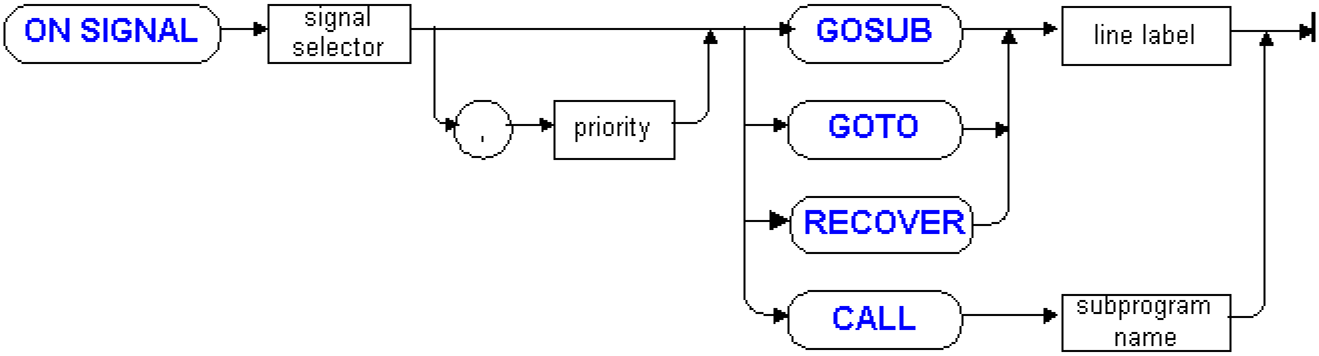
ENABLE, DISABLE, OFF KNOB, KNOBX, KNOBY, SYSTEM PRIORITY

ON SIGNAL

Defines an event branch for SIGNAL statement.

Syntax: ON SIGNAL signal-number [,priority] action

where: action = { GOTO|GOSUB|RECOVER } line | CALL subprogram
line = line-number | line-label



Sample:
```
ON SIGNAL Selector,Priority CALL Sub2
ON SIGNAL RECOVER Trap
ON SIGNAL 8 GOTO 770
```

View Sample: ON SIGNAL.BAS **(also found in examples directory)**

Description:

ON SIGNAL enables an event branch which occurs when a SIGNAL statement is executed using the same signal-number. The signal-number is a numeric expression rounded to an integer with a range of zero through fifteen.

Executing **ON SIGNAL** while another **ON SIGNAL** is still in effect for that same signal number overrides the previous **ON SIGNAL** definition. If the **ON SIGNAL** is executed in a different program context the original **ON SIGNAL** definition is restored when control returns to the calling context.

ON SIGNAL is canceled by OFF SIGNAL and disabled by DISABLE. A SUBEXIT, SUBEND, or RETURN from the defining subprogram also cancels it.

More information about **ON SIGNAL** can be found under the "Common Information" heading of the ON CYCLE manual entry.

See Also:

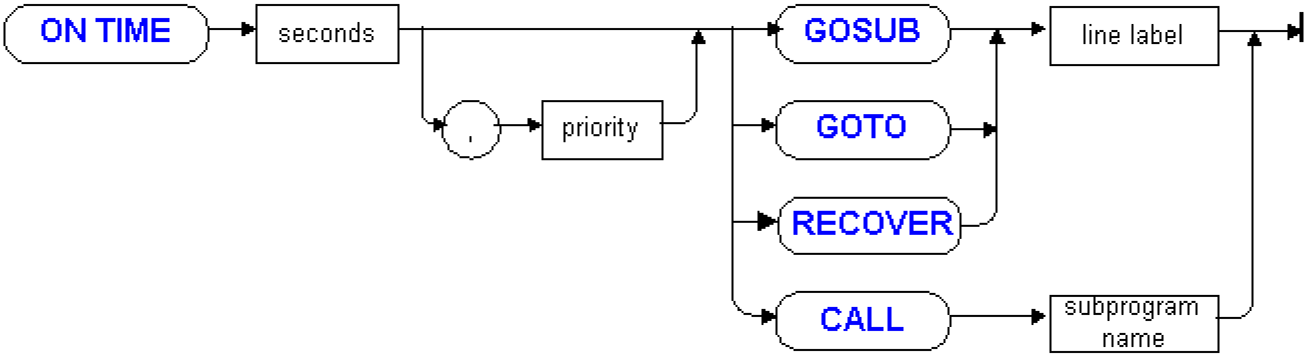
ENABLE, DISABLE, OFF SIGNAL, SIGNAL, SYSTEM PRIORITY

ON TIME

Defines a single event branch for a specific time.

Syntax: ON TIME time [,priority] action

where: time = numeric expression in range 0 to 86,399.99.
action = { GOTO|GOSUB|RECOVER } line | CALL subprogram
line = line-number | line-label



Sample: `ON TIME Hour*3600,T_pri CALL Explode`
`ON TIME (TIMEDATE+3600) MOD 86400 GOTO 2000`

View Sample: **ON TIME.BAS** **(also found in examples directory)**

Description:

ON TIME defines an event branch to occur when the real-time-clock reaches a specified time. The time is specified as the number of seconds since midnight. The time specified is rounded to the resolution of the computer clock.

There is only one **TIME** timer. Executing a new **ON TIME** while another **ON TIME** is still in effect will cause the **TIME** timer to use the new value. If the **ON TIME** is executed in a different program context, the original **ON TIME** definition is restored when control returns to the calling context. The old **TIME** value is not restored, however.

ON TIME is canceled by OFF TIME and disabled by DISABLE. A SUBEXIT, SUBEND, or RETURN from the defining subprogram also cancels it.

More information about **ON TIME** can be found under the "Common Information" heading of the ON CYCLE manual entry.

See Also:

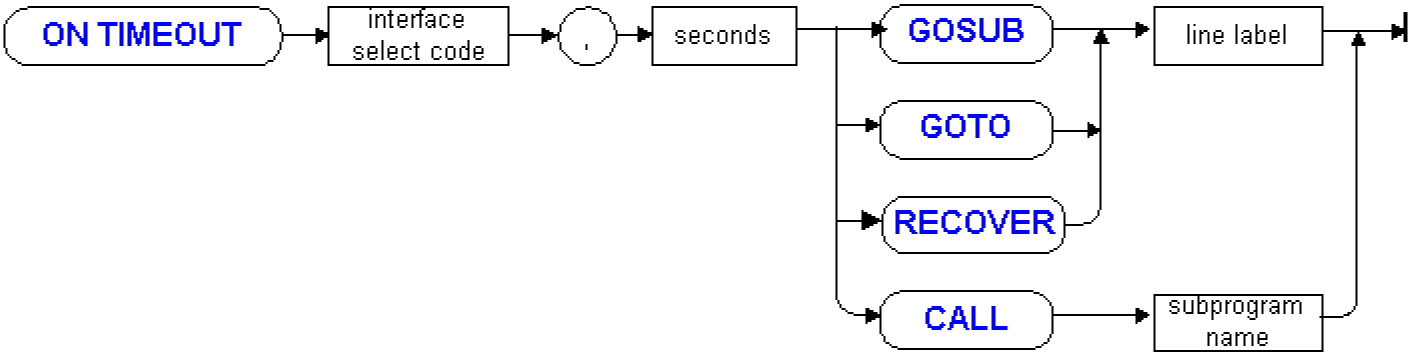
ENABLE, DISABLE, OFF TIME, SYSTEM PRIORITY, TIME$, TIMEDATE

ON TIMEOUT

Defines an event branch for an I/O timeout.

Syntax: ON TIMEOUT interface-select-code, seconds action

where: action = { GOTO | GOSUB | RECOVER } line | CALL subprogram
LINE = line-number | line-label



Sample:

```
ON TIMEOUT 4,5 GOTO 2000
ON TIMEOUT Printer,Sec GOSUB Message
ON TIMEOUT 4,1/2 RECOVER Line
```

View Sample: ON TIMEOUT.BAS **(also found in examples directory)**

Description:

ON TIMEOUT defines an event branch to take when an I/O operation on the specified interface fails to responded within the specified number of seconds. The value of seconds can range from 0.001 to 32.767 but is rounded to the timing resolution of the computer. The **ON TIMEOUT** statement must be executed before the I/O statement. If an **ON TIMEOUT** is not specified for a particular interface and a device does not respond to an I/O action, the computer will wait forever. Pressing the CLR I/O key will abort such an infinite wait.

TIMEOUTs work with the ENTER, OUTPUT, PRINTALL IS, PRINTER IS and PLOTTER IS statements, but not with the CONTROL, STATUS, READIO or WRITEIO statements or with the CRT or KBD interfaces or with files.

ON TIMEOUT is canceled by OFF TIMEOUT but is not disabled by DISABLE. A SUBEXIT, SUBEND, or RETURN from the defining subprogram also cancels it.

When returning from a CALL or GOSUB,execution continues with the line following the line causing the timeout.

More information about **ON TIMEOUT** can be found under the "Common Information" heading of the ON END manual entry.

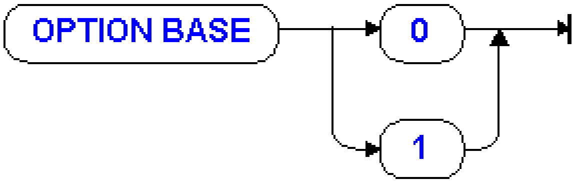
See Also:

OFF TIMEOUT, ON END, ON ERROR

OPTION BASE

Sets the default lower bound of array subscripts.

Syntax: OPTION BASE {0 | 1}



Sample:
```
OPTION BASE 0
OPTION BASE 1
```

View Sample: **OPTION BASE.BAS** **(also found in examples directory)**

Description:

The default array subscript lower bound may be specified in each program context with the **OPTION BASE**statement. It must appear in the program context before any COM, COMPLEX, DIM, INTEGER or REAL statements. There may be only one **OPTION BASE** statement in any program context. If there is no **OPTION BASE** statement then the default lower bound is zero.

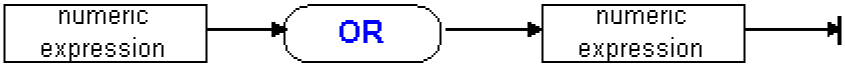
See Also:

BASE, COM, DIM, INTEGER, REAL

OR

Returns the logical inclusive OR of two expressions.

Syntax: numeric-expression OR numeric-expression



Sample:

```
A=1 OR 0
IF ProcA OR ProcB THEN Next
IF A=B OR X>Y THEN 1000
X=N+4*(J=1 OR K=2)
```

View Sample: **OR.BAS** **(also found in examples directory)**

Description:

The result of A **OR** B is zero only if both A and B are zero. If either or both A and B are non-zero, the result is one.

See Also:

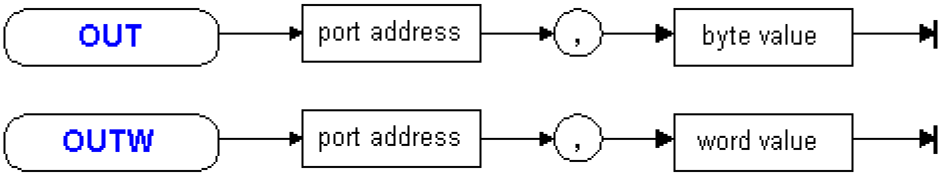
AND, NOT, EXOR

OUT and OUTW

Outputs a byte or word to an I/O Port.

Syntax: OUT port-address, byte-value
OUTW port-address, word-value

where: port-address = numeric-expression rounded to an integer
byte-value = numeric-expression rounded to an integer in the range 0 to 255
word-value = numeric-expression rounded to an integer



Sample:

```
OUT &H300,64+16
OUTW Base+3,&HF001
```

View Sample: **OUT.BAS** **(also found in examples directory)**
View Sample: **OUTW.BAS** **(also found in examples directory)**

Description:

The **OUT** statement outputs a byte to the specified I/O port. It is equivalent to WRITEIO 8080,Port;Byte. The **OUTW** statement outputs a word to the specified I/O port. It is equivalent to WRITEIO -8080,Port;Word. These statements are useful for doing I/O with devices, data acquisition boards, etc. for which there is no device driver available.

Some operating systems, such as Windows NT protect I/O ports; applications are not allowed to read or write to them. Under such operating systems, these functions are not allowed.

Porting to HP BASIC:

OUT and **OUTW** are new HTBasic statements that are not available in HP BASIC. They should not be used in programs that must be ported back to HP BASIC.

See Also:

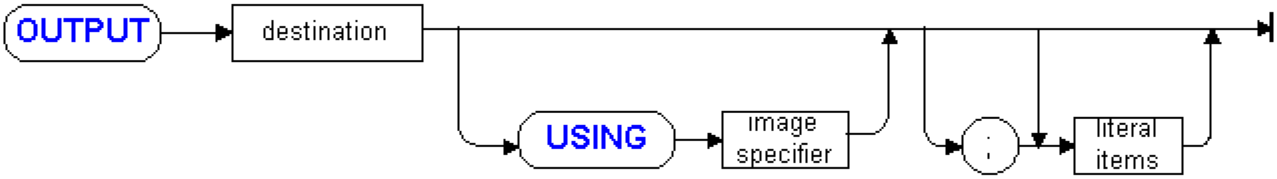
INP and INPW, READIO, WRITEIO

OUTPUT

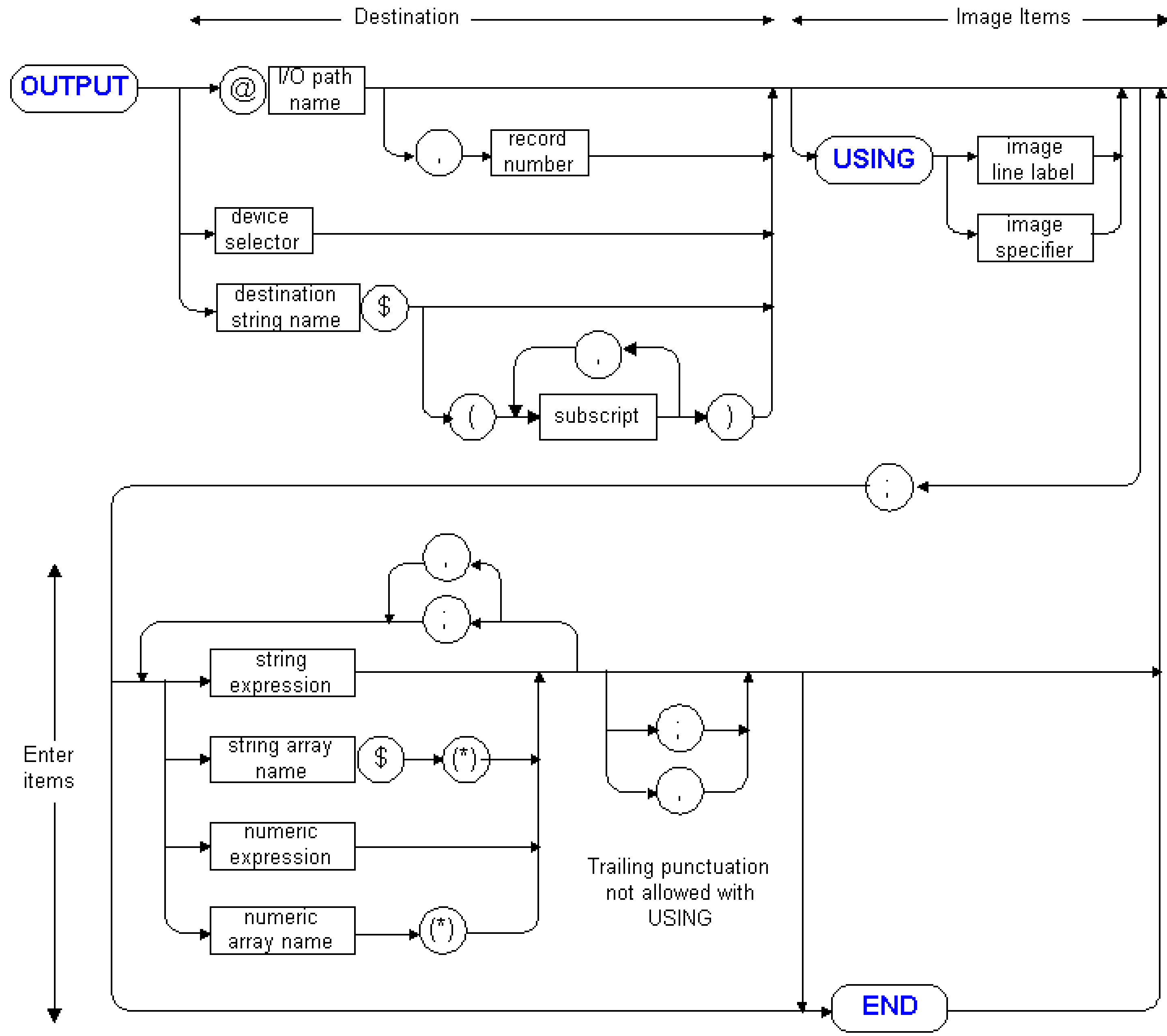
Outputs items to a specified destination.

Syntax: OUTPUT dest [USING image] [; items [{,|;}] [END]]

where: dest = @io-path [,record-number] |
device-selector |
string-name$ [(subscripts)]
items = item [{,|;} item [{,|;} item...]]
item = numeric-expression | numeric-array(*) |
string-expression | string-array$(*)
image = line-number | line label | string-expression
See IMAGE for image syntax.
subscripts = subscript [,subscript...]



expanded diagram:



Sample: `OUTPUT @Test;Sarray(*)`

```
OUTPUT @Sequence,4 USING SpecA;Part(3)
OUTPUT 10 USING "6A";V$[2;6]
OUTPUT @Printer;Order;SSN;Work$,END
```

View Sample: OUTPUT.BAS **(also found in examples directory)**

Description:

Numeric data, array elements or character strings are output to the specified destination.

Unless USING is specified, numeric items are output in standard numeric format. If the absolute value is in the range 1E-4 to 1E+6, it is rounded to twelve digits and output in floating point form. Otherwise the number is output in scientific notation.

Full arrays are output in row major order, using the full array specifier, "(*)". Each element is an item and is separated by a comma or semicolon if one follows the array name.

Destinations

File. An ASCII, BDAT or ordinary file may be used as the destination. The file must have been ASSIGNed to an I/O path. The ASSIGN statement determines the attributes to be used. With FORMAT ON, BDAT and ordinary files are written as ASCII characters. With FORMAT OFF, BDAT and ordinary files are written in internal format (explained below). An ASCII file is always written as ASCII characters. All files may be accessed serially and additionally, BDAT and ordinary files may be accessed randomly by including a record number.

String. A string may be used as the destination. **OUTPUT** begins at the beginning of the string and writes it serially.

Device. A device-selector or I/O path may be used to **OUTPUT** items to a device. The default system attributes are used with a device-selector. The ASSIGN statement determines the attributes used with an I/O path.

If the device selector is one, then the destination is the CRT. If the device selector is two, then the destination is the keyboard. This can be used to enter the keyboard function key sequences into the keyboard buffer. Each function sequence is two bytes, a CHR$(255) followed by the function specifier.

Buffer. A buffer assigned to an I/O path may be used as the destination. The buffer fill pointer points to the buffer location to be written next and is updated as data is **OUTPUT**. If the empty pointer is encountered, an error is generated.

FORMAT

If the FORMAT ON attribute is specified in the ASSIGN statement, the output is sent in ASCII format and the punctuation following each item affects the output. A semicolon causes an item to be sent with nothing following it, a comma causes a string item to be sent with a CR/LF following it and a numeric item to be sent with a comma following it. If no punctuation follows the last **OUTPUT** item, the EOL sequence follows it and if punctuation follows the last **OUTPUT** item, the EOL sequence is not output.

A complex number is output in rectangular form, real part first, then a comma and finally, the imaginary part. If a semicolon follows the complex item then the comma is not output.

If the FORMAT OFF attribute is specified in the ASSIGN statement the output is sent in internal format (explained below) and the punctuation following each item has no effect on the output.

END

The optional **END** may be used after the last data item. If USING is not specified, then **END**: 1) suppresses the EOL sequence from being output after the last item, 2) sends an EOI signal with the last character of the last item sent to a IEEE-488 device, and 3) truncates a file.

If USING is specified, then **END**: 1) suppresses the EOL sequence only when no data is output from the last output item, 2) sends EOI with the last character of the last item (unless no data is sent from the last item) and 3) truncates a file. A comma before **END** will output an item terminator (a comma for numeric items or a CR/LF for string items).

USING

See IMAGE for a complete explanation of the image list. The items specified in the image list are acted upon as they are encountered. Each image list item should have a matching output item. Processing of the image list stops when no matching output item is found. Conversely, the image list is reused starting at the beginning to provide matches for all remaining output items. FORMAT ON is used in connection with **OUTPUT USING**, even if FORMAT OFF has been specified.

OUTPUT USING is not allowed to ASCII files. Use **BDAT** or ordinary files or if necessary, do the

OUTPUT USING to a string and then **OUTPUT** the string to the ASCII file.

Internal Format (FORMAT OFF)

The internal format for an INTEGER is a two byte, two's complement, binary integer. LSB/MSB FIRST (see ASSIGN) can be used to specify the order in which the two bytes are sent or received. Internally, the order is stored in the form most natural to the computer's processor.

The internal format for REAL numbers is an eight byte, IEEE compatible floating point number (see *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std. 754-1985). As with integers, LSB/MSB FIRST can be used to determine the byte ordering during I/O statements.

A COMPLEX number is stored internally as two real numbers.

The internal format for strings depends on the source/destination of the I/O statement. The string format for devices and **BDAT** files consists of a string length followed by the string contents. Specifically, a four byte integer is sent/received first. The integer specifies the length of the string. The actual string is then sent/received. An even number of bytes is always sent/received, therefore, if the string is odd in length an extra padding byte is sent/received. As with integers, LSB/MSB FIRST can be used to determine the byte ordering of the integer length.

For ordinary files, the internal format for strings is a null-terminated string. For ASCII files FORMAT ON/OFF has no affect. Data is always stored as ASCII strings proceeded by a two byte length and padded by a space if necessary to make the string length even. The string length is always stored with MSB FIRST.

Records

When outputing to a file, you may specify a record number. The first record in the file is record 1. The record size for **BDAT** files is specified when the file is created and defaults to 256 bytes. For other file types the record size is 1; thus the record number is actually the offset into the file. The first byte of the file is at offset 1. When a record number is specified and the record size is not 1, if the **OUTPUT** produces more data than a single record, an End of Record error or event occurs.

OUTPUT KBD Porting

Three editor functions have been added to HTBasic and should not be used in programs that will be executed with HP BASIC: DEL LEFT, NEXT WORD and PREV WORD. Otherwise, all the two-character function key sequences (CHR$(255)&CHR$(X)) used by HP BASIC are compatible with HTBasic. If multiple statements are output in a single **OUTPUT** KBD statement, they are all executed before the next BASIC line. HP BASIC sometimes intermixes the execution with multiple BASIC lines, based on the presence or absence of "closure keys."

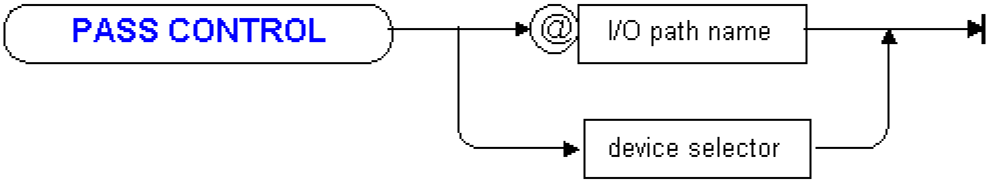
See Also:

ASSIGN, ENTER, IMAGE, INPUT, PRINT

PASS CONTROL

Passes Active Controller capability.

Syntax: PASS CONTROL {@io-path | device-selector}



Sample: `PASS CONTROL 719`
`PASS CONTROL @Dev`

View Sample: PASS CONTROL.BAS **(also found in examples directory)**

Description:

If an IO-path is specified, it must be assigned to a IEEE-488 device. If the computer is the active controller and a primary address is specified, control is passed to the addressed device. An error is generated if the computer is not the active controller or only an interface select code is specified. The specified device is talk addressed, a Take-Control-Message (TCT) is sent and the Attention line is set false. The computer then becomes a bus device, as opposed to a bus controller.

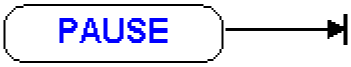
See Also:

ABORT , CLEAR, LOCAL, PPOLL, REMOTE, REQUEST, SEND, SPOLL, TRIGGER

PAUSE

Pauses program execution.

Syntax: PAUSE



Sample: `PAUSE`

View Sample: **PAUSE.BAS** **(also found in examples directory)**

Description:

PAUSE stops program execution before the next program line. The values of the variables in the current program context may be examined and modified. The CONTINUE key or the CONT command will resume program execution. RUN must be used to restart program execution if a program is modified during **PAUSE**.

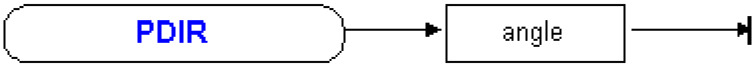
See Also:

CONT, TRACE

PDIR

Sets the rotation angle for IPLOT, RPLOT, POLYGON and RECTANGLE.

Syntax: PDIR angle



Sample:

```
PDIR 45
IF Ready THEN PDIR Graphangle
```

View Sample: **PDIR.BAS** **(also found in examples directory)**

Description:

The angle is a numeric-expression that specifies the direction and amount of rotation. It is measured in a counter-clockwise direction from the positive X-axis. Rotation is about the local point of origin. The current trigonometric mode (RAD or DEG) determines the units for angle. The default mode is RAD.

See Also:

DEG, IPLOT, LDIR, PIVOT, POLYGON, RAD, RECTANGLE, RPLOT

PEN

Sets the line color or physical pen.

Syntax: PEN pen-number

Sample:

```
PEN 3
PEN -1
PEN Feltpen
```

View Sample: **PEN.BAS** **(also found in examples directory)**

Description:

The **PEN** statement sets the color which will be used for line drawing. The pen can also be changed with PLOT, IPLOT, RPLOT and SYMBOL arguments. See COLOR for a complete explanation of pen-numbers for the CRT.

For a plotter, the **PEN** statement selects one of the available pens. The **PEN** number is sent to the plotter without any range checking. You should specify only values that are legal on your plotter. Note that for HPGL plotters, a pen number of zero instructs the plotter to put away the pen.

Drawing Mode Table

The writing mode of the pen is specified by the current drawing mode and the sign of the pen number. GESCAPECRT,4 is used to change to normal drawing mode. GESCAPE CRT,5 is used to change to alternate drawing mode. The following table defines the different writing modes available. P is a positive pen number, X is the present value of a pixel.

	GESCAPE CRT,4	**GESCAPE CRT,5**
Statement	**Normal**	**Alternate**
PEN P	P	BINIOR(X,P)
PEN 0	BINCMP(X)*	0
PEN -P	BINAND(X,BINCMP(P))	BINEOR(X,P)

*PEN 0 in Normal Drawing Mode will do BINCMP(X) in non-color map mode and 0 in COLOR MAP mode.

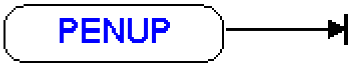
See Also:

AREA PEN, COLOR, SET PEN

PENUP

Raises the PEN on the current plotting device.

Syntax: PENUP



Sample: `PENUP`

View Sample: **PENUP.BAS** **(also found in examples directory)**

Description:

Raises the PEN on the current plotting device.

See Also:

CLIP, SHOW, VIEWPORT, WINDOW

PERMIT

Changes file protection permissions.

Syntax: PERMIT specifier [; protection [; protection...]]

where:

specifier = file-specifier | path-specifier
protection = category : [permission [,permission...]]
category = OWNER | GROUP | OTHER
permission = READ | WRITE | SEARCH

Sample:

```
PERMIT "/home/anita";OWNER : READ,WRITE,SEARCH
PERMIT "/dir/file";GROUP : READ;OTHER : READ;OWNER : READ
PERMIT "file2";OTHER :;GROUP :
```

Description:

On operating systems which support file permissions, **PERMIT** changes the permissions assigned to a file. If the operating system does not support this feature or does not support some of the categories or codes you specify or if you do not have the proper privilege to change the permissions, an error is returned.

This statement is not supported by HTBasic. Use PROTECT instead.

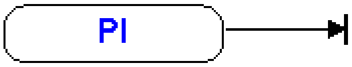
See Also:

CAT, CHOWN, CHGRP, CREATE, PROTECT, TIMEZONE IS

PI

Returns the value 3.14159265358979.

Syntax: PI



Sample:

```
Theta=PI
Area=PI*Radius^2
```

View Sample: **PI.BAS** **(also found in examples directory)**

Description:

The function **PI** returns an approximation of the value of the mathematical constant *Pi*, which is the ratio of the circumference of a circle to its diameter.

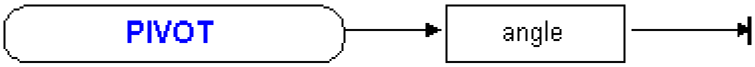
See Also:

ACS, ASN, ATN, COS, DEG, RAD, SIN, TAN

PIVOT

Rotates the coordinates of all drawn lines.

Syntax: PIVOT angle



Sample:

```
PIVOT 90
IF Adjust THEN PIVOT Lines
```

View Sample: **PIVOT.BAS** **(also found in examples directory)**

Description:

Angle is a numeric-expression that specifies the amount of rotation for all subsequently drawn lines. The rotation is done about the logical pen position when the **PIVOT** statement is executed. Positive values rotate counter-clockwise. Non-zero values of **PIVOT** cause the physical and logical pen positions to be different. Logical pen movement is unaffected. LABELs and AXES statements are unaffected.

The current trigonometric mode (RAD or DEG) determines the units for angle. The default mode is RAD.

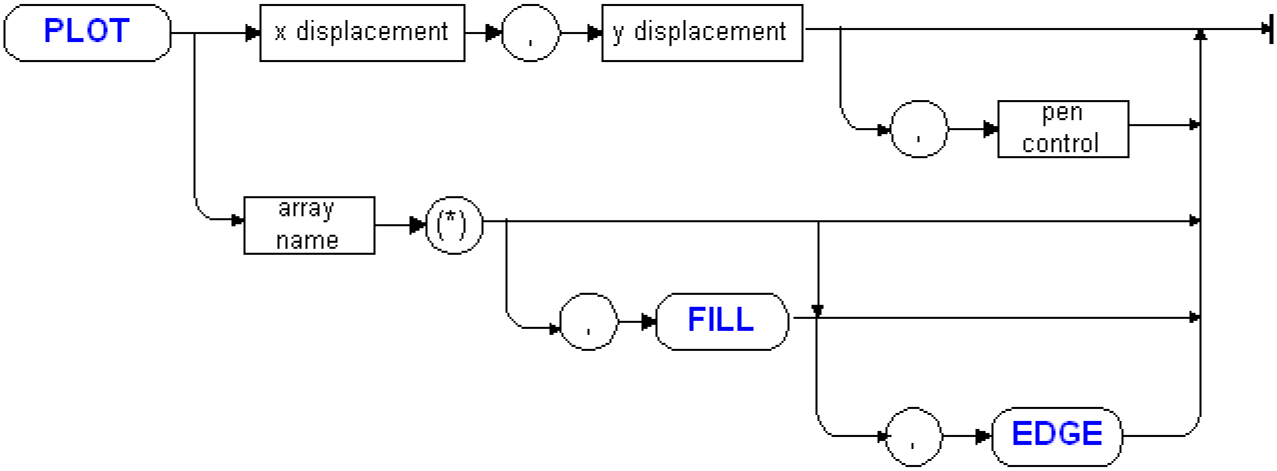
See Also:

DEG, IPLOT, LDIR, PDIR, PLOT, POLYGON, POLYLINE, RAD, RECTANGLE, RPLOT

PLOT

Moves the pen to the specified X and Y coordinates.

Syntax: PLOT x-position, y-position [,pen-control]
PLOT numeric-array(*) [,FILL] [,EDGE]



Sample: `PLOT 25,50`

```
PLOT Xx,Yy,Pen
PLOT Array(*)
PLOT Picto(*),FILL,EDGE
```

View Sample: **PLOT.BAS** **(also found in examples directory)**

Description:

The **PLOT** statement moves the pen to the specified X and Y position. You may specify when the pen is to be raised or lowered with the optional pen-control value. A two or three column array may be used to supply the coordinate and pen-control values.

If you specify a destination which is outside the clipping area, the logical position is set to that point but the pen is not moved. Only the portion of the vector which lies inside the clipping area is plotted.

The PIVOT statement affects the **PLOT** statement.

Pen-control

The optional pen-control value controls whether the pen is moved up or down and whether the change occurs before or after the move:

Pen-control Value	Affect
zero and positive even	raise after move
positive odd	lowered after move
negative odd	lowered before move
negative even	raised before move

The default pen-control value, one, specifies the pen is lowered after a move.

Array

PLOT uses a two-dimensional two- or three-column array to plot polygons. The array specifies the polygon shape using column one for X coordinates and column two for Y coordinates. The optional third-column specifies the operation (pen-control, AREA PEN, AREA INTENSITY, LINE TYPE, PEN, **FILL** and **EDGE**) for each row of the array. If a two-column array is specified, the default operation on each row is one, pen down after move.

The table below shows the meaning of columns 1 and 2 for each of the operations specified in column 3. These operations apply to **PLOT**, IPLOT, RPLOT and SYMBOL.

Column 1	Column 2	Column 3	Column 3 Meaning
X value	Y value	< -2	use even/odd pen control
X	Y	-2	Pen up before moving
X	Y	-1	Pen down before moving
X	Y	0	Pen up after moving
X	Y	1	Pen down after moving
X	Y	2	Pen up after moving
pen number	---	3	PEN
line type	repeat value	4	LINE TYPE
color	---	5	AREA INTENSITY
---	---	6	Start polygon mode w/FILL
---	---	7	End polygon mode
---	---	8	End of data for array
---	---	9	No operation, values ignored
---	---	10	Start polygon w/EDGE
---	---	11	Start polygon w/FILL & EDGE
---	---	12	Draw a FRAME
pen number	---	13	AREA PEN
red value	green value	14	AREA INTENSITY
blue value	---	15	AREA INTENSITY
---	---	> 15	No operation, values ignored

Select AREA R/G/B color

Operation 5 in column 3 selects the AREA INTENSITY color (see COLOR for an explanation of AREA INTENSITY colors). The column one value is divided into red, green and blue numbers, each five bits in length (the sixteenth bit of column one is ignored). Each five-bit number specifies a value in the range zero to sixteen. This number is subtracted from sixteen to calculate the intensity value for each of the colors: red, green, blue. Intensities range in value from zero (darkest) to sixteen (most intense).

For example, if column 1 is set to zero, then each of the three groups in column 1 is set to zero. Sixteen minus zero yields sixteen for all three groups. Sixteen is full intensity, therefore, the area fill color will be white.

The following equation calculates the value for column one given R, G, B values in the range zero to one.

Column1 = 16-16*R + SHIFT(16-16*G,-5) + SHIFT(16-16*B,-10)

Operations 14 and 15 can also be used to select the AREA INTENSITY red, green and blue values. The range of intensity is zero (no color) to 32,767 (full intensity). Operation 14 should be done before 15 and the operation takes effect when operation 15 is done.

FILL and EDGE

A polygon is formed from a line sequence of 2 or more points with the optional **FILL** or **EDGE** specifiers. A polygon is drawn by plotting the first point, each successive point and closed by drawing the final point back to the first point.

If **FILL** is specified, the polygon is filled with the current AREA fill color and if **EDGE** is specified, the polygon is edged with the current PEN color. The array pen-control instructions supersede any other instructions on pen movement, LINE TYPE , **FILL,** and **EDGE** specifiers.

See Also:

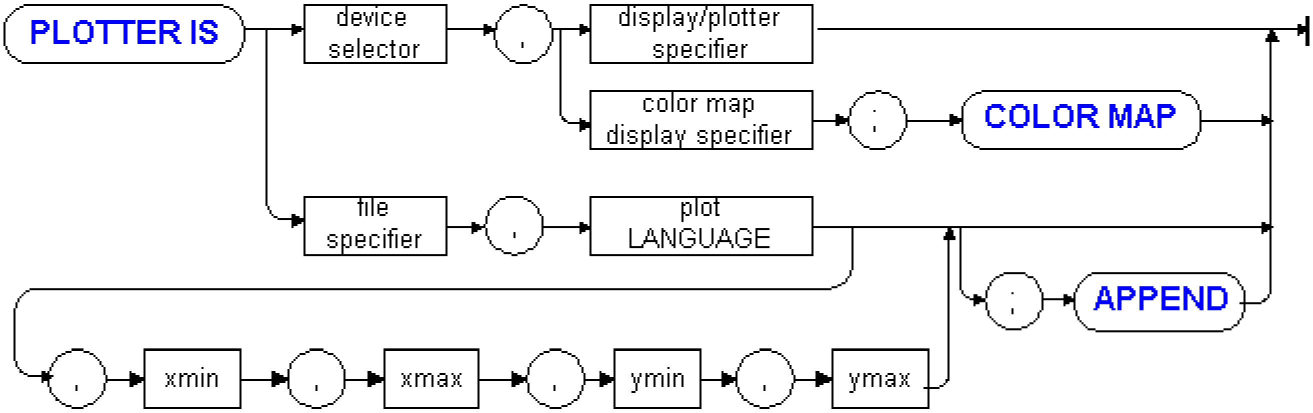
AREA, CLIP, DRAW, IPLOT, MOVE, POLYLINE, POLYGON, RPLOT

PLOTTER IS

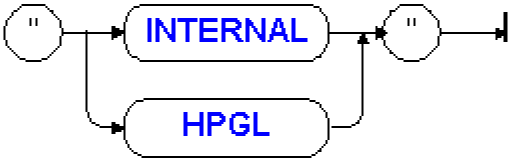
Specifies the graphics output device and language.

Syntax: PLOTTER IS destination, language [,hard-clip]
[; { APPEND|COLOR MAP }]

where: destination = file-specifier | device-selector
language = string expression which resolves to the name
of a graphics driver and can include driver options
hard-clip = xmin,xmax,ymin,ymax - four numeric-expressions specifying
the size of the drawing surface



literal form of display specifier/plotter specifier/color map display specifier:



Sample:
```
PLOTTER IS CRT,"INTERNAL";COLOR MAP
PLOTTER IS 10,"HPGL",2,268,0,190
PLOTTER IS 26,"PS",2,268,0,190
PLOTTER IS "Pictfile","HPGL",5.75,250.50,7.25,136.875
PLOTTER IS CRT,"INTERNAL"
```

View Sample: **PLOTTER IS.BAS** **(also found in examples directory)**

Description:

The **PLOTTER IS** statement directs vector graphics to a device or file. (Use the DUMP DEVICE IS statement to print bit-mapped graphics from the screen to a device or file.) The default **PLOTTER IS** device is the CRT. Executing a **PLOTTER IS** statement directs all subsequent graphics output to the specified target.

The destination of the **PLOTTER IS** statement tells the graphic driver where to send output. Output can go to the display, device or file, although not every driver can send output to all the targets. For example, display drivers can only send output to the display and it doesn't make sense to send GIF output to anything but a file.

Display

To direct output to the CRT, use the reserved word CRT as the destination or the interface select codes 1, 3 or 6. For most display drivers, the value affects how the driver handles text as explained below.

Devices

To specify a device such as a plotter or a printer capable of vector graphics, use the interface select code of the interface connecting the device. Use the device-selector if the device is on the IEEE-488 bus. If hard-clip limits are specified, they are given in the order "xmin, xmax, ymin, ymax" and are specified in millimeters. If the hard-clip limits are not specified, they are read from the device when this statement is executed. The specified device must respond to this query or the computer will wait indefinitely for the response. Use the CLR-I/O key to stop the computer if it gets stuck in this state.

The following example sends HPGL commands to a LaserJet III printer. The first line resets the printer, starts landscape printing and switches into HPGL mode. The second line directs plotter output to LPT1 and sets the hard-clip units for an 8-1/2 x 11 sheet of paper:

```
OUTPUT 26;CHR$(27)&"E"&CHR$(27)&"&l1O"&CHR$(27)&"%1B";
PLOTTER IS 26,"HPGL",2,268,0,190
```

Files

To send graphics output to a file, the target should be replaced with the file name. The file must be an existing ordinary or BDAT file. The hard-clip limits may be specified or defaulted to ±392.75 mm in the x axis and ±251.5 mm in the y axis. The hard-clip limits are "xmin, xmax, ymin, ymax" and are specified in millimeters. If **APPEND** is not specified, the file is positioned to the beginning and truncated. The file is closed when another **PLOTTER IS**, GINIT or SCRATCH A statement is executed. Example:

```
CREATE "DRAW.PLT",0
PLOTTER IS "DRAW.PLT","HPGL"
```

Language

HTBasic supports loadable graphics drivers. The language string expression specifies the name of a driver. The first time a driver is specified in a **PLOTTER IS** statement, the driver is loaded and graphics are directed to it. When the plotting language is subsequently specified, the driver is not loaded again, but graphics are again directed to it. The following table lists the drivers available at the time of this manual printing. (Not all drivers are available in all versions.)

Name	**Type**	**Display Adapter**
INTERNAL	CRT	Reuse last CRT driver specified
WIN	CRT	Windows Display Driver
HPGL	Graphic	Hewlett-Packard Graphic Language
PS	Graphic	PostScript printers, plotters, and files

"INTERNAL" is a special language string synonymous with the last CRT specified. The following examples illustrate use of the **PLOTTER IS** statement in selecting device drivers:

```
PLOTTER IS CRT,"INTERNAL"
PLOTTER IS 712,"HPGL"
PLOTTER IS "file.ps";"PS"
```

HTBasic automatically loads the WIN display driver when it starts. It is recommended that **PLOTTER IS** statements be included in your AUTOST file to load any necessary drivers. (Drivers can also be loaded in immediate mode when the BASIC RUNLIGHT is Idle.)

Driver Options

It is sometimes necessary to specify options for the graphic drivers. Options are included by appending a semicolon to the driver name, followed by the options. The syntax for specifying options in the **PLOTTER IS** statement is:

PLOTTER IS target,"driver[;options]"

The specific driver sections contain more details on these options.

WIN Driver

The WIN driver is a CRT driver that uses the Microsoft Windows display drivers.

For compatibility with HP BASIC/UX, options for the WIN driver are specified on the command line. Command line switches were explained in Chapter 1. These command line switches are passed to the WIN driver:

Switch	**Effect**
-colors	Number of Colors to Use
-fn	Use named font
-geometry	Specify initial size of HTBasic window
-title	Specify the window title

Window Resize

Resizing the HTBasic window using the mouse is supported, but has the following effects. If the number of text columns changes, any text present is discarded. If in edit mode, the screen is redrawn using the new size.

Any graphics present in the window are discarded. The current pen position is left undefined. The VIEWPORT, WINDOW and hard clip limits are unchanged, although GESCAPECRT,3 returns the new window size. Use the GINIT statement to set the VIEWPORT, WINDOW and hard clip limits to the new window size. Or use the

```
PLOTTER IS CRT,"INTERNAL"
```

statement to activate use of the new hard clip limits without the side effects of GINIT.

HPGL Driver

The HPGL plotter driver generates HP-GL language plots from HTBasic plotting commands. The driver supports most variations of HP-GL, including HP-GL/2 and the printer form of HP-GL/2 included in PCL-5. The HPGL2 plotter driver is loaded with a line like

PLOTTER IS *device*,"HPGL[;*options*]",[*p1x,p2x,p1y,p2y*]

or

PLOTTER IS "*file*","HPGL[;*options*]",[*p1x,p2x,p1y,p2y*]

In the above, *device* refers to an HTBasic device number. *File* refers to a file in the computer's file system. The file must already exist when the PLOTTER IS statement is executed.

Plotting Area

The points *(p1x,p1y)* and *(p2x,p2y)* determine the lower left and the upper right corners of a rectangular area the driver will plot to. These points are specified in mm from the lower left corner of the paper. *P2x* and *p2y* must be larger than *p1x* and *p1y*, respectively. All of these coordinates must be positive or zero if the PCL5 option is used (see Options, below). If the plotting area is omitted, the driver reads the plot area from the plotter, if it is connected to a serial or IEEE-488 port. If output is directed to a file, the driver uses the default values from the table below.

PCL5 Option	**Orientation**	**(P1x,P1y)**	**(P2x,P2y)**
No	Landscape	(-393, -252)	(393, 252)
No	Portrait	(-252, -393)	(252, 393)
Yes	Landscape	(0,0)	(254, 184)
Yes	Portrait	(0,0)	(184, 254)

Options

The options are listed after the semicolon in the driver name, within the quotes. If more than one option is specified, the option names are separated by commas. When no options are specified, the HPGL2 driver produces the same output as the HPGL driver. A table in the *Installing and Using* manual may help in choosing from the options.   The options are as follows:

COLOR. This option tells the driver that the device used for plotting is a color printer with plotter functions, such as the Hewlett-Packard DeskJet 1600C. This option is ignored unless the PCL5 option is also specified.

FILL. This option tells the driver that the plotter being used can do area filling. Area filling produced by the plotter is generally much faster than that produced by the driver.

GRAY. This option causes the driver to produce grayscale plots when used with a printer. Each color that normally would be plotted is changed to a brightness using the method explained in the Pen Colors section, below, before plotting. Note that the brightness level is inverted unless the **INVERT** option is also used. The **GRAY** option need not be specified; it is the default. This option is ignored unless the PCL5 option is also used.

HPGL2. By default, the driver produces plots for an HP-GL plotter. This option allows the driver to produce plots for an HP-GL/2 plotter, such as the Hewlett-Packard DraftMaster. Since HP-GL/2 plotters can all do area filling, the **HPGL2** option turns on the **FILL** option.

INVERT. By default, the driver reverses black and white on color plots and reverses all gray levels on grayscale plots when the plots are made on a printer. This is suitable for printers that use dark inks on white paper, but is the opposite of the colors normally shown on the computer screen. The **INVERT** option causes colors or gray levels to be represented as they are on the computer screen. This option is ignored unless the **PCL5** option is also used.

PCL5. This option tells the driver that the plotter is a laser or electrostatic printer with built-in plotter emulation using the PCL-5 language. This causes the driver to send escape sequences at the beginning and end of plots to enable and disable the plotter emulation. When this option is used, a **PLOTTER IS CRT**,"INTERNAL" statement should be executed at the end of plotting to make the printer eject the page containing the plot. Since all PCL-5 devices use the HP-GL/2 plotter language, this option turns on the HPGL2 and **FILL** options.

PORTRAIT. The **PORTRAIT** option causes the driver to produce plots in portrait orientation, that is, with the long edge of the paper vertical. Without this option, the driver produces plots in landscape orientation, with the long edge of the paper horizontal.

Polygons

The HPGL driver, for compatibility with HP BASIC, outputs polygon fills as separate lines. However, the driver can be instructed to output HPGL/2 polygon fill commands. This is useful if the plotter supports the polygon fill command or if an HPGL file is produced for import into another program that supports polygons. To enable polygon mode, use GESCAPE code 104, operation number 1:

```
10  INTEGER Param(1)
20  Param(0)=1    ! HPGL Operation Number 1 is HPGL/2 Flag
30  Param(1)=1    ! Set HPGL/2 Flag to 1=enable, 0=disable
40  GESCAPE Isc,104,Param(*)
```

If output is to a device, substitute the device ISC for Isc in line 40. If output is to a file, substitute 1 for Isc.

Pen Colors

When the HPGL2 driver is used with a pen plotter, the HTBasic PEN command selects the indicated pen on the plotter. However, when the driver is used with a printer (as indicated by the PCL5 option), the effect of the PEN command is that described in the following text.

The colors or grayscales produced by each pen depend on the states of the COLOR and **INVERT** options used in loading the driver, as well as the state of the **COLOR MAP** option of the HTBasic CRT driver. If the **COLOR MAP** option is off, the following gray levels or colors are used:

PEN	GRAY	COLOR	GRAY INVERT	COLOR, INVERT
0	white	white	black	black
1	black	black	white	white
2	30% black	red	70% black	red
3	89% black	yellow	21% black	yellow
4	59% black	green	41% black	green
5	70% black	cyan	30% black	cyan
6	11% black	blue	89% black	blue
7	40% black	violet	60% black	violet
8	black	black	white	white
9	30% black	red	70% black	red
10	89% black	yellow	21% black	yellow
11	59% black	green	41% black	green
12	70% black	cyan	30% black	cyan
13	11% black	blue	89% black	blue
14	40% black	violet	60% black	violet
15	black	black	white	white

If the **COLOR MAP** option of the CRT driver is on, the plot is made using the colors in the HTBasic color map if the COLOR option is used. If the **INVERT** option is not used, black and white are reversed. If the COLOR option is not used, the colors in the HTBasic color map are converted to shades of gray using the NTSC equation:

brightness = 11% blue + 59% green + 30% red

If the **INVERT** option is not used, the brightness is inverted before plotting is done. With both pen plotters and printers, the sign of the pen is ignored; the absolute value determines the pen used.

Drawing Mode

When the PCL5 option is specified, the HTBasic statement GESCAPECRT,5 sets alternate drawing mode for the driver. Normally, the driver replaces anything previously at a location with what is currently drawn. In the alternate drawing mode, the previous black or colored areas show through the white areas of the new plot. The HTBasic statement GESCAPECRT,4 returns the driver to normal drawing mode.

Line Thickness

If the PCL5 option is specified, line thicknesses can be set in the driver. Lines default to 0.35 mm thick. The line thickness for all pens can be changed by the GESCAPECRT,104 statement as in either of the examples below:

```
INTEGER Param(1:2)           ! an array for the command
Param(1) = 10                ! line thickness code
Param(2) = thickness         ! desired thickness (in 1/100 GDU's)
GESCAPE CRT,104,Param(*)     ! send thickness
```

```
INTEGER Param(1:2)           ! an array for the command
Param(1) = 11                ! line thickness code
Param(2) = thickness         ! desired thickness (in 1/100 mm)
GESCAPE CRT,104,Param(*)     ! send thickness
```

Line Caps and Joins

When the PCL5 option is specified, line cap and join styles can be specified. By default, the device driver uses round caps to end lines and round joins to connect lines, which simulates the round pens used on pen plotters. This can be changed with the following statements.

```
INTEGER Param(1:3)           ! an array for the command
Param(1) = 12                ! line thickness code
Param(2) = cap               ! desired line cap
Param(3) = join              ! desired line join
GESCAPE CRT,104,Param(*)     ! set cap and join
```

The values for *cap* and *join* can be selected from the following tables.

Cap	Meaning
1	butt cap
2	square cap
3	triangular cap
4	round cap

Join	Meaning
1	mitered join
2	mitered, beveled if too long
3	triangular join
4	round join
5	beveled join
6	no join

Note that many low-resolution PCL-5 devices use a butt cap and no join with lines less than 0.35 mm thick, regardless of the cap and join settings.

Crosshatching

The HPGL driver can crosshatch areas meant to be filled. This is its default behavior unless the FILL or PCL5 option is specified, in which case the default is to use solid fills.

If the FILL or PCL5 options are specified, the driver can be made to crosshatch filled areas with the following statements:

```
INTEGER Param(1:2)              ! an array for the command
Param(1) = 1                    ! set fill type
Param(2) = state                ! turn solid filling on or off
GESCAPE CRT,104,Param(*)        ! send command
```

State is 0 to use crosshatching and any other value to use solid filling. For compatibility with older drivers, if *state* is nonzero, this command turns on the FILL option if neither the FILL nor the PCL5 option was specified when the driver was loaded.

When crosshatching is turned on, the following sets of statements can be used to control the crosshatch parameters. If these statements are not executed, crosshatching is done with solid horizontal lines spaced 0.01 in. (0.25 mm) apart, which is useful on most devices for producing a solid fill.

```
INTEGER Param(1:2)              ! an array for the command
Param(1) = 2                    ! set crosshatch type
Param(2) = type
GESCAPE CRT,104,Param(*)        ! send command
```

Type is 1 for single hatching, 2 for crosshatching.

```
INTEGER Param(1:2)              ! an array for the command
Param(1) = 3                    ! set hatch angle
Param(2) = angle                ! desired angle, degrees
GESCAPE CRT,104,Param(*)        ! send command
```

Angle is the angle in degrees (regardless of the HTBasic RAD or DEG setting) for hatching. *Angle* is rounded to the nearest multiple of 45 degrees.

```
INTEGER Param(1:2)              ! an array for the command
Param(1) = 4                    ! set line spacing
Param(2) = spacing              ! desired spacing (in 1/100 GDU's)
GESCAPE CRT,104,Param(*)        ! send command
```

```
INTEGER Param(1:2)              ! an array for the command
Param(1) = 5                    ! set line spacing
Param(2) = spacing              ! desired spacing (in 1/100 mm)
GESCAPE CRT,104,Param(*)        ! send command
```

The above commands are equivalent except that in the first command, *spacing* is expressed in 1/100 GDU and in the second in 1/100 mm.

```
INTEGER Param(1:3)              ! an array for the command
Param(1) = 6                    ! set line type for hatching
Param(2) = type                 ! desired line type for crosshatching
Param(3) = size                 ! desired pattern repetition size
GESCAPE CRT,104,Param(*)        ! send command
```

Type is the type of line, as listed in the LINE TYPE section of the *HTBasic Online Reference Manual*. *Size* is the pattern repetition length in 1/100 GDU's. This would be 100 times the pattern repetition length specified in a LINE TYPE statement.

Pages

The GCLEAR statement causes subsequent plotting to be done on a new page. If the PCL5 option is specified, the GCLEAR statement causes the printer to eject the old plot. Also, opening a file with

```
PLOTTER IS "file","HPGL";APPEND
```

causes the driver to append new pages of plot information to the current file if it exists already. Note that most word processor programs and other programs that can import files will probably superimpose the plots imported from a file containing more than one plot.

Ending Plots

If the PCL5 option is used, the HPGL2 driver will not eject a plot until a GCLEAR statement is executed, HTBasic is ended, or when the PLOTTER IS device is set to a different device. It is recommended that a statement like

```
PLOTTER IS CRT,"INTERNAL"
```

be placed at the end of each program section that produces a plot using the PCL5 option driver.

PostScript Driver

The PostScript graphics output driver generates PostScript-language files from HTBasic plotting commands. These files are suitable for printing on PostScript-language printers and photographic equipment and for importing into documents using the PostScript file format. The PostScript graphics output driver is loaded with the following statement:

PLOTTER IS *destination*,"PS[;*options*]",[*p1x,p2x,p1y,p2y*]

Destination refers to a device or file. If it is a file, the file must already exist when the PLOTTER IS statement is executed and it should be an *ordinary file*. Otherwise the HTBasic file header will appear as bad data at the start of the file.

The points *(p1x,p1y)* and *(p2x,p2y)* determine the lower left and the upper right corners of a rectangular area the driver will plot to. These points are specified in mm from the lower left corner of the paper. All of these coordinates must be positive or zero and *p2x* and *p2y* must be larger than *p1x* and *p1y*, respectively. If omitted, the driver uses *(p1x,p1y)* = (25.4 mm, 25.4 mm) and *(p2x,p2y)* = (262.7 mm, 190.5 mm) in landscape mode and *(p2x,p2y)* = (190.5 mm, 262.7 mm) in portrait mode, which produces a plot with adequate margins on US "A" or European A4 size paper. Note that most PostScript printers cannot print to the edges of the paper. Because of this, the points specified should include a small (about 1 cm) margin on each side when the driver is used with a printer.

Options

The options are listed after the semicolon in the driver name, within the quotes. If more than one option is specified, the option names are separated by commas. The options are as follows:

COLOR. This option causes the driver to produce color plots. Note that black and white are inverted from their values on the screen unless the **INVERT** option is also used. Color plots require a PostScript level 2 output device or a PostScript level 1 device with color language extensions.

GRAY. This option causes the driver to produce grayscale plots. Each color that normally would be plotted is changed to a brightness using the method explained in the Pen Colors section, below, before plotting. Note that the brightness level is inverted unless the **INVERT** option is also used. The **GRAY** option need not be specified; it is the default.

INVERT. By default, the driver reverses black and white on color plots and reverses all gray levels on grayscale plots. This is suitable for printers that use dark inks on white paper, but is the opposite of the colors normally shown on the computer screen. The **INVERT** option causes colors or gray levels to be represented as they are on the computer screen.

PORTRAIT. The **PORTRAIT** option causes the driver to produce plots in portrait orientation, that is, with the long edge of the paper vertical. Without this option, the driver produces plots in landscape orientation, with the long edge of the paper horizontal.

Pen Colors

The colors or grayscales produced by each pen depend on the states of the COLOR and **INVERT** options used in loading the driver, as well as the state of the **COLOR MAP** option of the HTBasic CRT driver. If the **COLOR MAP** option is off, the following gray levels or colors are used:

PEN	GRAY	COLOR	GRAY INVERT	COLOR, INVERT
0	white	white	black	black
1	black	black	white	white
2	30% black	red	70% black	red
3	89% black	yellow	21% black	yellow
4	59% black	green	41% black	green
5	70% black	cyan	30% black	cyan
6	11% black	blue	89% black	blue
7	40% black	violet	60% black	violet
8	black	black	white	white
9	30% black	red	70% black	red
10	89% black	yellow	21% black	yellow
11	59% black	green	41% black	green
12	70% black	cyan	30% black	cyan
13	11% black	blue	89% black	blue
14	40% black	violet	60% black	violet
15	black	black	white	white

If the **COLOR MAP** option of the CRT driver is on, the plot is made using the colors in the HTBasic color map if the COLOR option is used. If the **INVERT** option is not used, black and white are reversed. If the COLOR option is not used, the colors in the HTBasic color map are converted to shades of gray using the HTSC equation:

brightness = 11% blue + 59% green + 30% red

If the **INVERT** option is not used, the brightness is inverted before plotting is done. GESCAPE codes 4 and 5 are ignored as is the sign of the PEN. Graphics always overwrite existing graphics.

Line Thickness

Lines default to 0.35 mm thick. The line thickness can be changed by the GESCAPECRT,104 statement as in either of the examples below:

```
INTEGER Param(1:2)             ! an array for the command
Param(1) = 10                  ! line thickness code
Param(2) = thickness           ! desired thickness (in 1/100 GDU's)
GESCAPE CRT,104,Param(*)       ! send thickness
```

```
INTEGER Param(1:2)             ! an array for the command
Param(1) = 11                  ! line thickness code
Param(2) = thickness           ! desired thickness (in 1/100 mm)
GESCAPE CRT,104,Param(*)       ! send thickness
```

Line Caps and Joins

By default, the device driver uses round caps to end lines and round joins to end lines, which simulates the round pens used on pen plotters. This can be changed with the following statements.

```
INTEGER Param(1:3)             ! an array for the command
Param(1) = 12                  ! set line cap and join
Param(2) = cap                 ! desired line cap
Param(3) = join                ! desired line join
GESCAPE CRT,104,Param(*)       ! set cap and join
```

The values for *cap* and *join* can be selected from the following tables.

Cap	**Meaning**	**Join**	**Meaning**
1	butt cap	1,2	mitered join, beveled if too long
2	square cap	3,4	round join
3,4	round cap	5,6	beveled join

Crosshatching

By default, the PostScript plotter driver fills areas with shades of gray or color (if the COLOR option has been specified). The driver can be made to crosshatch filled areas with the following statements.

```
INTEGER Param(1:2)             ! an array for the command
Param(1) = 1                   ! set fill type
Param(2) = state               ! turn solid filling on or off
GESCAPE CRT,104,Param(*)       ! send command
```

State is 0 to use crosshatching and any other value to use solid filling.

When crosshatching is turned on, the following sets of statements can be used to control the crosshatch parameters. If these statements are not executed, crosshatching is done with solid horizontal lines spaced 0.01 in. (0.4 mm) apart.

```
INTEGER Param(1:2)             ! an array for the command
Param(1) = 2                   ! set crosshatch type
Param(2) = type
GESCAPE CRT,104,Param(*)       ! send command
```

Type is 1 for single hatching, 2 for crosshatching.

```
INTEGER Param(1:2)             ! an array for the command
Param(1) = 3                   ! set crosshatch angle
Param(2) = angle               ! desired angle, degrees
GESCAPE CRT,104,Param(*)       ! send command
```

Angle is the angle in degrees (regardless of the HTBasic RAD or DEG setting) for hatching. *Angle* is rounded to the nearest integer.

```
INTEGER Param(1:2)             ! an array for the command
Param(1) = 4                   ! set line spacing
Param(2) = spacing             ! desired spacing (in 1/100 GDU's)
GESCAPE CRT,104,Param(*)       ! send command
```

```
INTEGER Param(1:2)             ! an array for the command
Param(1) = 5                   ! set line spacing
Param(2) = spacing             ! desired spacing (in 1/100 mm)
GESCAPE CRT,104,Param(*)       ! send command
```

The above commands are equivalent except that in the first command, spacing is expressed in 1/100 GDU and in the second in 1/100 mm.

```
INTEGER Param(1:3)             ! an array for the command
Param(1) = 6                   ! set line type for hatching
Param(2) = type                ! desired line type
Param(3) = size                ! desired pattern repetition size
GESCAPE CRT,104,Param(*)       ! send command
```

Type is the type of line, as listed under the LINE TYPE topic in the *HTBasic Reference Online Manual*. *Size* is the pattern repetition length in 1/100 GDU's. This would be 100 times the pattern repetition length specified in a LINE TYPE command.

Pages

The GCLEAR statement causes subsequent plotting to be done on a new page. The driver inserts a PostScript "%%Page" comment at the beginning of each page. The comments are used by some print spooling software. Also, opening a file with

```
PLOTTER IS "file","PS";APPEND
```

causes the driver to append new pages of plot information to the current file if it exists already. Since the driver doesn't know how many pages are already in the file, it begins its "%%Page" comments with page 1. This may cause problems with some print spooling software.

Ending Plots

The PostScript language requires information at the end of a plot to cause the plot to be printed. This information is output when the GCLEAR statement is executed, HTBasic is exited, or when the **PLOTTER IS** device is set to a different device. It is recommended that a statement like

```
PLOTTER IS CRT,"INTERNAL"
```

be placed at the end of each program section that produces a plot using the PostScript driver.

Driver Loading

Up to ten graphic and dump drivers can be loaded at a time. It is recommended that for each driver needed, a **PLOTTER IS** statement is included in your AUTOST file to load it.

Driver files can be loaded at any point. To find the driver file HTBasic takes the driver specified in the **PLOTTER IS** statement and performs several operations upon it to find the correct file. ".DW6" is appended to the name. Then the following locations are searched, in the specified order:

1. The directory containing the HTBasic executable.
2. The current directory.
3. The Windows system directory (such as \WINNT\SYSTEM32).
4. The Windows directory.
5. The directories listed in the PATH environment variable.

Porting Issues

Both HP BASIC and HTBasic do an implicit **PLOTTER IS** assignment for you if you attempt to use graphic statements before an explicit **PLOTTER IS**. The difference is that HTBasic does the implicit **PLOTTER IS** as soon as HTBasic is started and HP BASIC waits until the first graphic statement is executed. The only known effect of the different approaches is that under HP BASIC, a SYSTEM$("PLOTTER IS") returns "0" until the first graphic statement is executed and HTBasic returns the correct value anytime.

HP BASIC supports only "INTERNAL" and "HPGL" graphic languages. HTBasic supports loadable graphic device drivers so it is not limited to these two choices. HTBasic also allows clip-limits to be specified when output is directed to a device, allowing use of plotters or printers that are incapable of returning p-points. Do not use HTBasic extensions if you wish to execute the same program with HP BASIC.

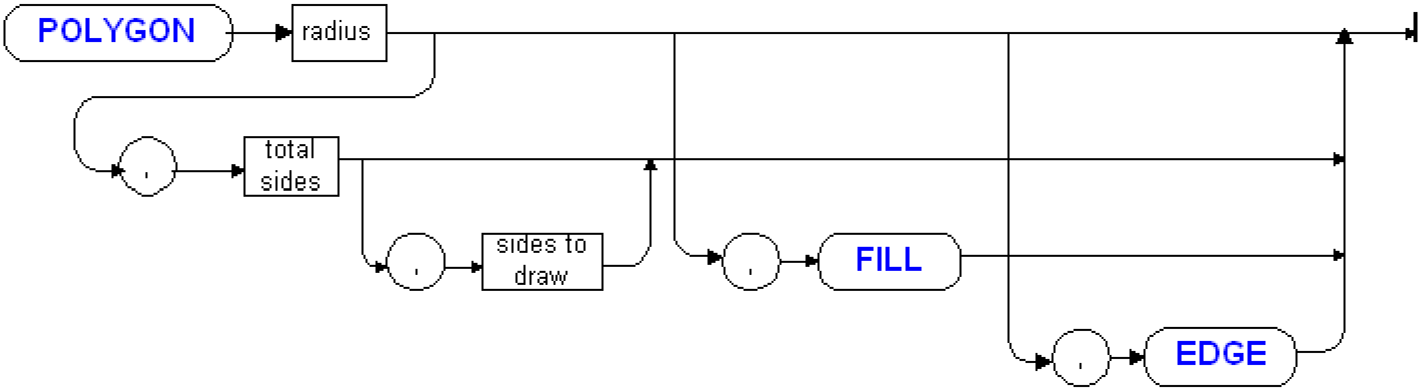
See Also:

COLOR, CONFIGURE DUMP, DUMP DEVICE IS, GRAPHICS INPUT IS, SET PEN

POLYGON

Draws a closed regular polygon, circle, or ellipse.

Syntax: POLYGON radius [,total-chords [,draw-chords]] [,FILL] [,EDGE]



Sample: `POLYGON Radius,Totside,Drawside`

```
POLYGON -Figure,7,FILL,EDGE
POLYGON 30,65,50
```

View Sample: **POLYGON.BAS** **(also found in examples directory)**

Description:

The **POLYGON** statement generates variable sided polygons or circles. The pen starts and ends a **POLYGON** execution in the same position and after execution the pen is up. The radius is the distance between the logical pen position and the polygon vertices where the first vertex is in the positive X axis direction. A negative radius will rotate the **POLYGON** 180 degrees.

The total number of chords is rounded to an integer and must be in the range 3 to 32,767. If not specified, sixty chords are drawn.

The optional number of chords to draw is rounded to an integer and must be in the range of one to 32,767. If not specified all chords are drawn.

If the number of chords drawn are less than the specified total number of chords, the polygon closure is affected. If the pen is up when the **POLYGON** statement is executed, the polygon is closed by drawing the last vertex to the first vertex. If pen is down, the polygon is closed by drawing the last vertex to the center of the polygon and then drawing from the center to the first vertex.

The polygon can be filled with the current AREA color and edged with the current PEN color and LINE TYPE. If neither are specified **EDGE** is assumed.

The PIVOT statement affects the **POLYGON** statement.

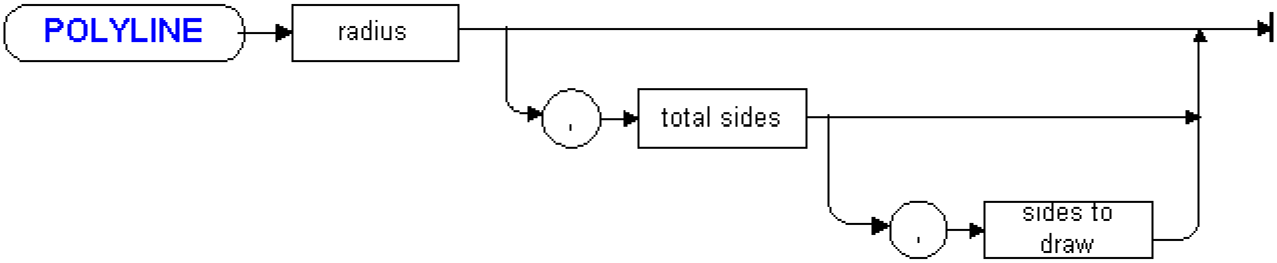
See Also:

MOVE, DRAW, PIVOT, PLOT, POLYLINE, RECTANGLE

POLYLINE

Draws an open regular polygon.

Syntax: POLYLINE radius [,total-chords [,draw chords]]



Sample:

```
POLYLINE 65,50,45
POLYLINE Radius,Chordtot,Chord
POLYLINE -Size,5
```

View Sample: **POLYLINE.BAS** **(also found in examples directory)**

Description:

The **POLYLINE** statement generates variable sided polygons or circles. The pen starts and ends a **POLYLINE** execution in the same position and after execution the pen is up.

The radius is the distance between the logical pen position and the polygon vertices where the first vertex is in the positive X-axis direction. A negative radius will rotate the POLYGON 180 degrees.

The total number of chords is rounded to an integer and must be in the range 3 to 32,767. If not specified, sixty chords are drawn.

The optional number of chords to draw is rounded to an integer and must be in the range of one to 32,767. If not specified all chords are drawn.

If the number of chords drawn are less than the specified total number of chords, the polygon is not closed. If the pen is up when the **POLYLINE** statement was executed, the first vertex is on the perimeter. If the pen is down when the **POLYLINE** statement was executed, the first point (logical pen position) is drawn to the first point on the perimeter.

The PIVOT statement affects the **POLYLINE** statement.

See Also:

MOVE, DRAW, PIVOT, PLOT, POLYGON, RECTANGLE

POS

Returns the position of one string within another.

Syntax: POS(search-string, match-string)

where: search-string and match-string = string-expressions

Sample:

```
I=POS(A$,B$)
IF POS(A$,B$(5)) THEN Start
P=POS(A$,"PN")
Hyphen=POS(Txt$,"-")
ON POS(Fk$,Key$) GOSUB 1000,2000,3000
```

View Sample: **POS.BAS** **(also found in examples directory)**

Description:

The **POS** function returns the character position in the search-string of a match-string. A value of zero is returned if the match-string is not found in the search-string or if the match-string has a zero length.

If a sub-string is specified for the search-string, the position returned is the position from the beginning of the sub-string not from the beginning of the full string.

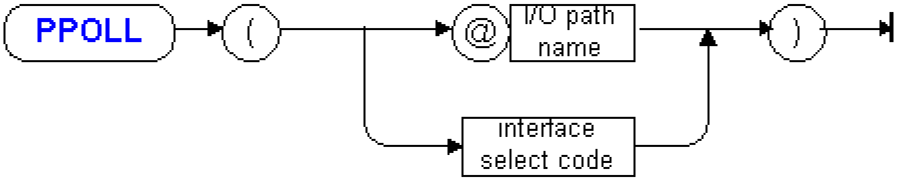
See Also:

CHR$, LWC$, NUM, REV$, RPT$, TRIM$, UPC$, VAL, VAL$

PPOLL

Conducts an IEEE-488 Parallel Poll and returns status.

Syntax: PPOLL({@io-path | interface-select-code})



Sample:

```
PPOLL(8)
PPOLL(@Gpib)
IF BIT (PPOLL(10),3) THEN Start
```

Description:

A IEEE-488 parallel poll is performed and an 8-bit status message from the IEEE-488 bus is returned. If the computer is not the active controller an error is generated. The I/O path or interface select code must refer to the IEEE-488 interface.

The bus action is as follows: ATN and EOI are set for ≥25 microsec., one byte of data is read from the bus, EOI is released, and ATN is restored to its previous state.

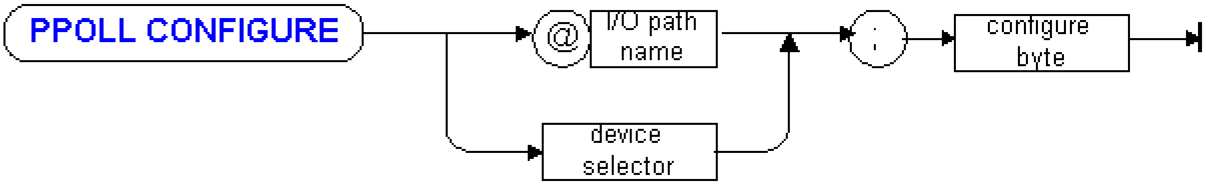
See Also:

ABORT , CLEAR, LOCAL, PASS CONTROL, REMOTE, REQUEST, SEND, SPOLL, TRIGGER

PPOLL CONFIGURE

Configures remote IEEE-488 device parallel poll response.

Syntax: PPOLL CONFIGURE {@io-path | device-selector} ; configure-byte



Sample:

```
PPOLL CONFIGURE 701;1
PPOLL CONFIGURE 702;3
PPOLL CONFIGURE @Dev;Sense
```

Description:

The device specified by the I/O path or the device selector is configured for a parallel poll response. If the computer is not the active controller an error is generated. The I/O path or device selector must refer to one or more IEEE-488 devices.

The configure byte is a numeric-expression rounded to an integer in the range zero to fifteen. The three least significant bits of its binary representation select the data bus line and the fourth bit selects the logical sense of the response.

The bus action is as follows: ATN, MTA, UNL, LAG, PPC, PPE.

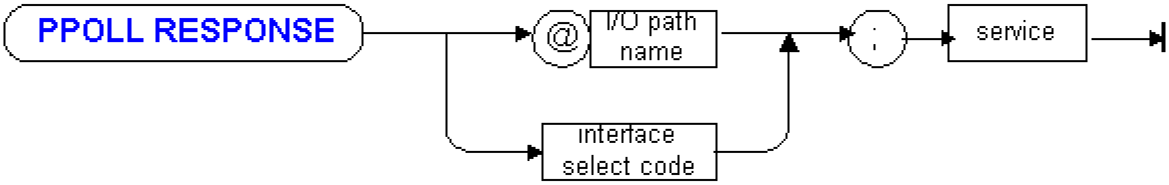
See Also:

ABORT , CLEAR, LOCAL, PASS CONTROL, PPOLL, REMOTE, REQUEST, SEND, SPOLL, TRIGGER

PPOLL RESPONSE

Configures local IEEE-488 device parallel poll response.

Syntax: PPOLL RESPONSE {@io-path | interface-select-code} ; service



Sample: `PPOLL RESPONSE Isc;Answer`
`PPOLL RESPONSE @Gpib;1`

Description:

This statement enables or disables this device to respond to a parallel poll request from the IEEE-488 bus active controller. If an I/O path is specified, it must refer to the IEEE-488 interface. A service value of zero disables the parallel poll response, whereas a value of one enables the parallel poll response. The device must be configured for a parallel poll response with the PPOLL CONFIGURE command. It specifies which bus data bit to respond on and the logical sense of the response.

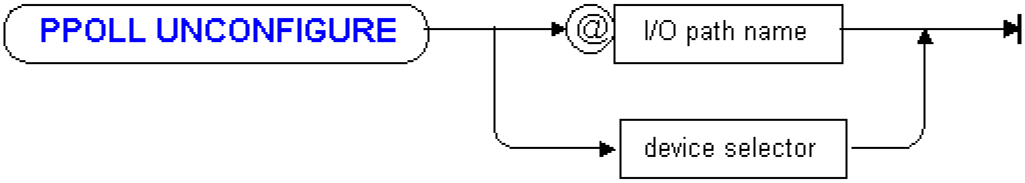
See Also:

ABORT , CLEAR, LOCAL, PASS CONTROL, PPOLL, REMOTE, REQUEST, SEND, SPOLL, TRIGGER

PPOLL UNCONFIGURE

Disables the parallel poll response of a specified device or devices.

Syntax: PPOLL UNCONFIGURE {@io-path | device-selector}



Sample: `PPOLL UNCONFIGURE 5`
`PPOLL UNCONFIGURE @Dev`

Description:

The device specified by the I/O path or the device selector is unconfigured for a parallel poll response. If the computer is not the active controller an error is generated. The I/O path or device selector must refer to one or more IEEE-488 devices.

If a primary device address is specified the bus action is: ATN, MTA, UNL, LAG, PPC, PPD; otherwise the bus action is: ATN, PPU.

See Also:

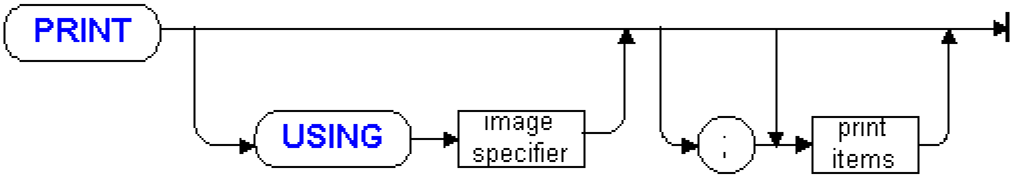
CLEAR, LOCAL, PASS CONTROL, PPOLL, REMOTE, REQUEST, SEND, SPOLL, TRIGGER

PRINT

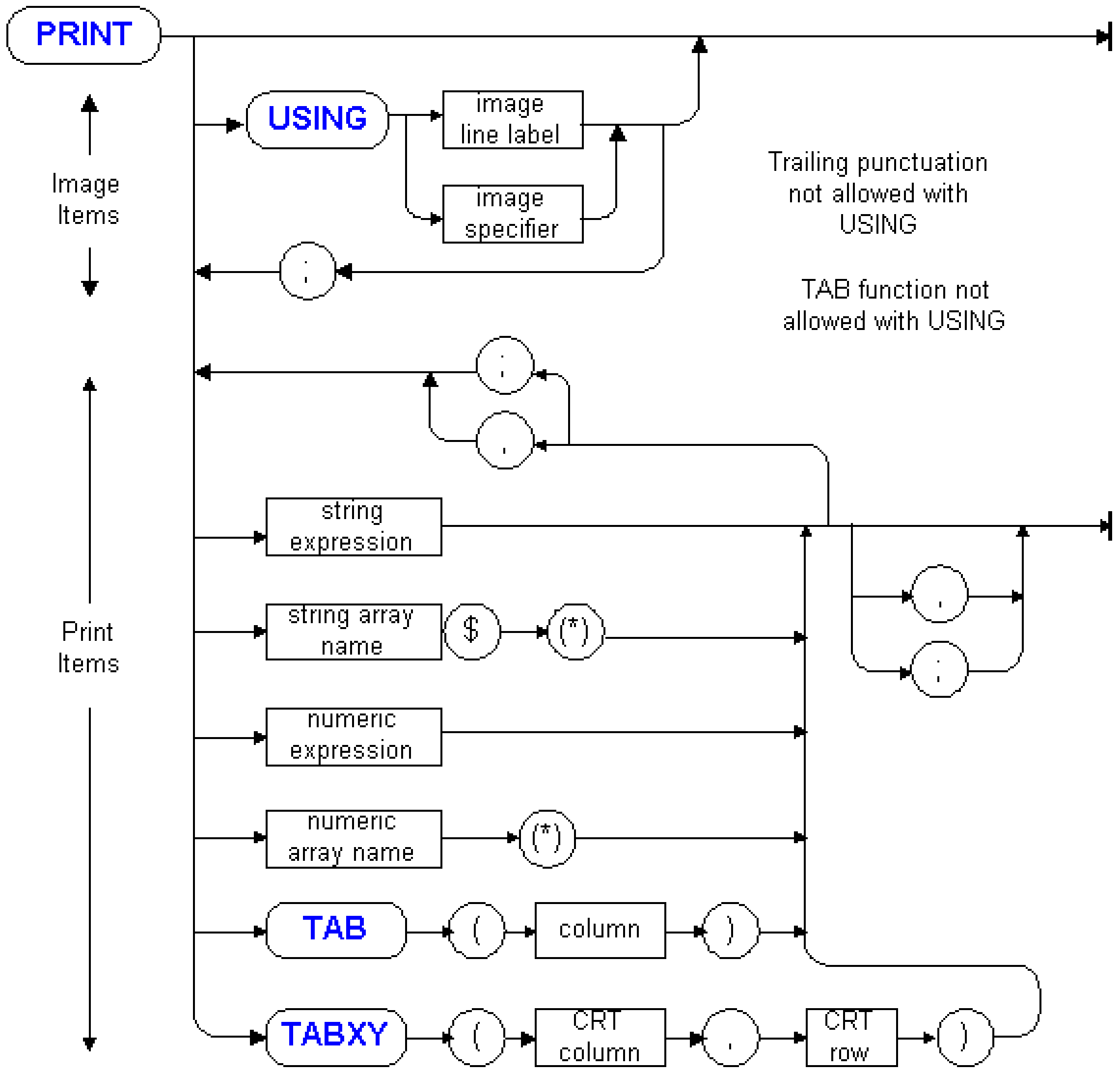
Outputs data to the PRINTER IS device.

Syntax: PRINT [items [{,|;}]]
PRINT USING image [;items]

where: items = item [{,|;} item [{,|;} item...]]
item = numeric-expression | numeric-array(*) |
string-expression | string-array$(*) |
TAB(crt-column) | TABXY(crt-column,crt-row)
image = line-number | line label | string-expression
See IMAGE for image syntax



expanded diagram:



Sample:

```
PRINT "Test Number ";N;
PRINT Values(*)
PRINT String$[1,8],TAB(12),Result
PRINT TABXY(1,1),Title$,TABXY(Col,3),Par$
PRINT USING 1040;R1,R2,R3
PRINT USING Fmt;Ssn,Item$,Weight
```

View Sample: PRINT.BAS **(also found in examples directory)**

View Sample: TAB.BAS **(also found in examples directory)**

View Sample: TABXY.BAS **(also found in examples directory)**

Description:

PRINT sends numeric data, array elements or character strings to the PRINTER IS device. The default PRINTER IS device is the CRT. The output may optionally be formatted with the USING image.

Unless **USING** is specified, numeric items are printed in standard numeric format. If the absolute value is in the range 1E-4 to 1E+6, it is rounded to twelve digits and printed in floating point form. Otherwise the number is printed in scientific notation.

If **USING** is not specified, then the punctuation following the item determines the item's print field width and suppresses the automatic **EOL** sequence. The compact field is used if a semicolon follows the item; and the default print field is used if a comma follows the item.

In both compact and default print form, numeric numbers are printed with one leading blank for positive numbers or the minus sign for negative numbers. In compact field form numeric items are printed with one trailing blank and string items are printed with no leading or trailing blanks. The default print field form prints items with trailing blanks to fill to the beginning of the next ten character field.

A complex number is printed in rectangular form, first the real part, then an extra space and finally the imaginary part.

Arrays

A full array may be printed in row-major order using the full array specifier, "(*)". If a semi-colon follows an array then the array elements are printed in compact fields. If a comma follows an array then default print fields are used. Additionally the automatic **EOL** sequence will be suppressed if either a semi-colon or a comma is used.

TAB/TABXY

The **TAB** function positions the next print character on the print line using the following equation: **TAB** column_position = ((column - 1) MOD screenwidth) + 1. The **TABXY** function positions the next print character on the CRT with X (column) and Y (row) coordinates. **TABXY**(1,1) specifies the upper-left of the CRT. A zero value for either argument specifies the current value for that argument.

If the CRT is not the PRINTER IS device, **TABXY** is ignored. **TAB** and **TABXY** can not be used with **USING**.

End-Of-Line

At the end of the list of items to **PRINT**, an **EOL** is sent to the PRINTER IS device. This can be suppressed by using trailing punctuation. **EOL** is also sent when the print position reaches the **WIDTH** of the printer. **WIDTH** and the **EOL** characters can be defined with the PRINTER IS command. The default **WIDTH** is the width of the screen or window, and the default **EOL** is CR/LF (CHR$(13) & CHR$(10)).

Control Characters

The following control characters have a special meaning when used in **PRINT** statements when the CRT is the PRINTER IS device:

Character	Meaning
CHR$(7)	Ring the bell.
CHR$(8)	Moves print cursor back one space.
CHR$(10)	Moves print cursor down one line.
CHR$(12)	Prints two line-feeds, scrolls output area buffer so next item goes to the top of the CRT.
CHR$(13)	Moves print cursor to column one.

Character	Meaning
CHR$(128)	All enhancements off.
CHR$(129)	Inverse mode on.
CHR$(130)	Blinking mode on.
CHR$(131)	Inverse and Blinking modes on.
CHR$(132)	Underline mode on.
CHR$(133)	Underline and Inverse modes on.
CHR$(134)	Underline and Blinking modes on.
CHR$(135)	Underline, Inverse, & Blinking modes on.

Character	Meaning
CHR$(136)	White
CHR$(137)	Red
CHR$(138)	Yellow
CHR$(139)	Green
CHR$(140)	Cyan
CHR$(141)	Blue
CHR$(142)	Magenta
CHR$(143)	Black

All other characters less than CHR$(32) are ignored. To print, rather than ignore, the characters in this range, use DISPLAY FUNCTIONS.

If some characters don't display correctly when you use the **PRINT** or LIST commands, it may be caused by conflicts with the attribute control characters in the range of 128 to 143. To move the attribute control characters from the range 128 to 143 down to the range 16 to 31, use the following command:

```
CONTROL CRT,100;1
```

With USING

See IMAGE for a complete explanation of the image list. The items specified in the image list are acted upon as they are encountered. Each image list item should have a matching print item. Processing of the image list stops when no matching print item is found. Conversely, the image list is reused starting at the beginning to provide matches for all remaining print items. FORMAT ON is used in connection with **PRINT USING**, even if FORMAT OFF has been specified.

Porting to HP BASIC:

CONTROL CRT, 100 is a new HTBasic feature that is not available in HP BASIC. It should not be used in programs that must be ported back to HP BASIC.

See Also:

ALPHA, IMAGE, INPUT, OUTPUT, READ

PRINT LABEL

Assigns a name to a data storage volume.

Syntax: PRINT LABEL volume-label [TO volume-specifier]

Sample:

```
PRINT LABEL "Officevol" TO "A:"
PRINT LABEL Vlabel$ TO Vol$
```

Description:

The volume label string is written to the specified device as the new label, overriding any previous volume label. This command is not supported by HTBasic. Use the OS LABEL command instead. The following example labels the floppy disk in drive A:

```
EXECUTE "LABEL A: WORKDISK"
```

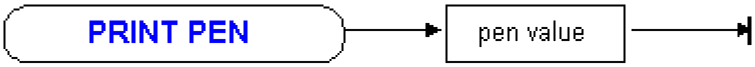
See Also:

CAT, COPY, CREATE, INITIALIZE, MASS STORAGE IS, PROTECT, PURGE, READ LABEL, RENAME, SYSTEM$("MSI")

PRINT PEN

Selects the pen color used for the output area and DISP line.

Syntax: PRINT PEN pen-number



Sample:

```
PRINT PEN Value
PRINT PEN 1
IF Green THEN PRINT PEN 2
```

View Sample: **PRINT PEN.BAS** **(also found in examples directory)**

Description:

This statement overrides any ALPHA PEN statement that may be in effect. The pen-number is a numeric expression rounded to an integer. If you are using CRTB, the bit-mapped display driver mode, legal values are from 0 to 15. (HP BASIC supports values to 255.) If you are using CRTA, the non-bit-mapped display driver mode, legal values are from 136 to 143. This statement is equivalent to CONTROL CRT,15;pen-number.

See Also:

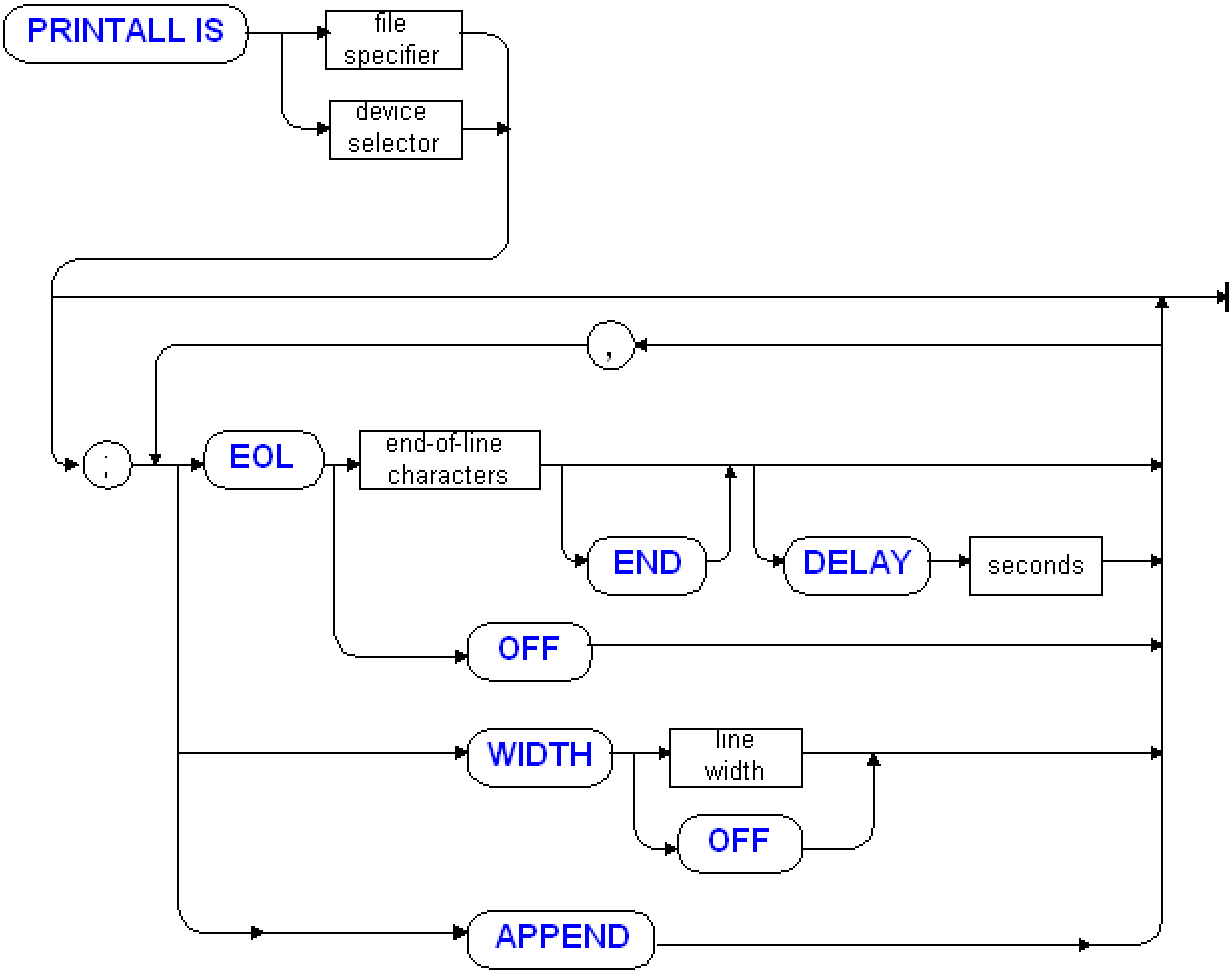
COLOR, ALPHA PEN, KBD LINE PEN, KEY LABELS PEN

PRINTALL IS

Assigns a logging device for operator interaction and error messages.

Syntax: PRINTALL IS destination [;attributes]

where: destination = device-selector | file-specifier
attributes = attribute [,attribute ...]
attribute = WIDTH {OFF|line-width} |
EOL end-of-line [END] [DELAY seconds] | OFF |
APPEND
end-of-line = string-expression, evaluating to a string
of eight characters or less.
seconds = numeric-expression, rounded to the timing
precision of the computer clock
line-width = numeric-expression, rounded to an integer



Sample:

```
PRINTALL IS Centronix
PRINTALL IS PRT;EOL CHR$(10) & CHR$(13) DELAY .5
PRINTALL IS Dev;WIDTH 120,EOL A$ END
```

View Sample: **PRINTALL IS.BAS** **(also found in examples directory)**

Description:

PRINTALL IS defines where to send output from print-all mode. When print-all mode is on, all messages output to the screen (including output area, DISP line, keyboard line and message line) are also output to the **PRINTALL** device. When print-all mode is off, output appears only in the normal places, and no information is sent to the **PRINTALL** target. The **PRINTALL** device is the CRT after start-up and SCRATCH A.

The print-all mode is toggled between on and off each time the PRT ALL key is pressed. STATUS(KBD,1) returns a 1 if print-all mode is on and 0 if it is off. A program can turn print-all mode on with CONTROL KBD,1;1 and off with CONTROL KBD,1;0.

Print-all is a powerful debugging tool. Use it in connection with TRACE to print TRACE messages about program execution. Also, certain error conditions can produce more than one line of output. Only the last message is visible on the message line. With print-all on, all the messages can be read on the **PRINTALL** device.

Destinations

The output can be sent to a device (usually a printer) or file. If the destination is a file, it must be an existing ordinary file or a BDAT file.

Sent to a printer, **PRINTALL** allows permanent logging of output.

Attributes

The **EOL** attribute specifies a new end-of-line string of up to eight characters. The **END** attribute specifies an **EOL** to be sent with the last character of the **EOL** string. The **DELAY** attribute specifies a time to wait after sending the **EOL** string and before continuing with program execution. The delay is in seconds and should be in the range 0.001 to 32.767 but is rounded to the timing resolution of the computer. The **OFF** attribute returns the **EOL** string to the default CR/LF, no **EOL** and no **DELAY**.

The **WIDTH** attribute specifies the maximum number of characters sent to the printing device before an automatic **EOL** sequence is sent. If **WIDTH OFF** is specified, the width is set to infinity. **WIDTH OFF** is the default.

If **APPEND** is specified and output is to a file, the file position is moved to the end-of-file before any data is sent to the file. If **APPEND** is not specified, the file contents are replaced with new data.

See Also:

CAUSE ERROR, CLEAR ERROR, ERRL, ERRLN, ERRM$, ERRN, ERROR RETURN, ERROR SUBEXIT, TRACE, XREF

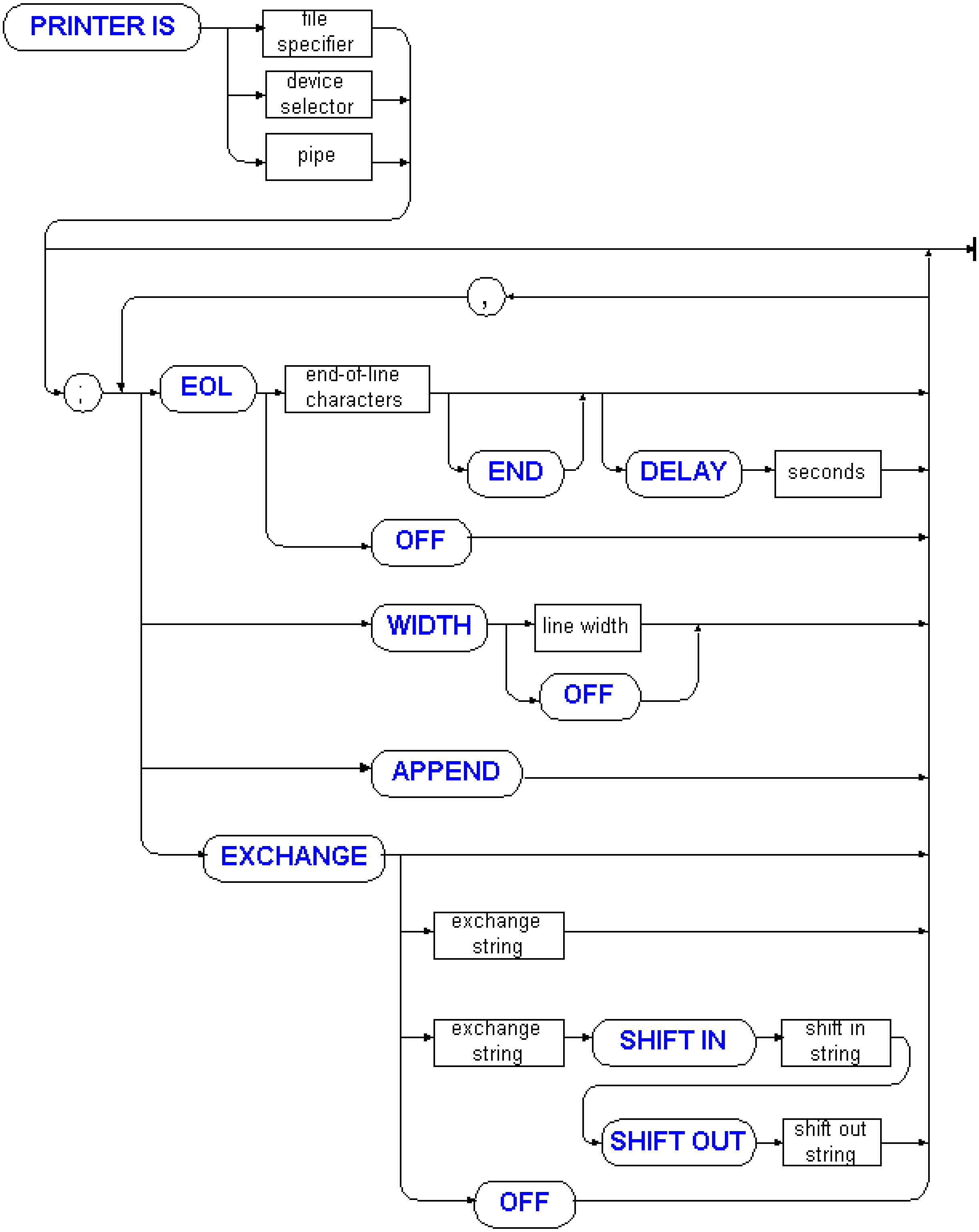
PRINTER IS

Specifies the system printing device.

Syntax: PRINTER IS destination [;attributes]

where:

destination = device-selector | file-specifier
attributes = attribute [,attribute ...]
attribute = WIDTH {OFF|line-width} | EOL end-of-line [END] [DELAY seconds] | OFF | APPEND
end-of-line = string-expression, evaluating to a string of eight characters or less.
seconds = numeric-expression, rounded to the timing precision of the computer clock
line-width = numeric-expression, rounded to an integer



Sample: `PRINTER IS 701`

```
PRINTER IS "Myfile";WIDTH 80
PRINTER IS 12;EOL A$ DELAY .5
PRINTER IS Dev;WIDTH 120,EOL My$ END
```

View Sample: PRINTER IS.BAS **(also found in examples directory)**

Description:

PRINTER IS specifies the destination for all PRINT, CAT and LIST statements which do not specify a destination. The **PRINTER** device is the CRT at start-up and after SCRATCH A.

Destinations

The output can be sent to a device (usually a printer) or a file. If the destination is a file, it must be an existing ordinary file or a BDAT file. If a file is specified, it is positioned to the beginning (unless **APPEND** is specified) and closed when another **PRINTER IS** or SCRATCH A statement is executed.

Attributes

The **EOL** attribute specifies a new end-of-line string of up to eight characters. The **END** attribute specifies an **EOL** to be sent with the last character of the **EOL** string. The **DELAY** attribute specifies a time to wait after sending the **EOL** string and before continuing with program execution. The delay is in seconds and should be in the range 0.001 to 32.767, but is rounded to the timing resolution of the computer. The **OFF** attribute returns the **EOL** string to the default CR/LF, no **EOL** and no **DELAY**.

The **WIDTH** attribute specifies the maximum number of characters sent to the printing device before an automatic **EOL** sequence is sent. If **WIDTH OFF** is specified, the width is set to infinity. If **WIDTH** is not specified, it defaults to the width of the screen.

If **APPEND** is specified and output is to a file, the file position is moved to the end-of-file before any data is sent to the file. If **APPEND** is not specified, the file contents are replaced with new data.

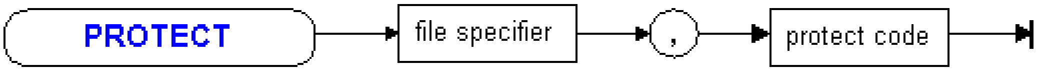
See Also:

CAT, IMAGE, LIST, PRINT

PROTECT

Changes file attributes.

Syntax: PROTECT file-specifier,protect-code



Sample:

```
PROTECT Mine$,"H"
PROTECT Name$,"R"
```

View Sample: PROTECT.BAS **(also found in examples directory)**

Description:

The **PROTECT** command differs from HP BASIC's **PROTECT** command. Under operating systems, like DOS, which do not support file passwords, the protect code is an operating system dependent string giving the file protections to be assigned to the file.

PROTECT is used to set file attributes. Three attributes are supported: read-only, system and hidden. The protect-code should be a numeric expression which contains zero, one or more of the characters "R", "S" and "H". Any attributes specified are turned on, any attributes not specified are turned off. For example:

```
PROTECT "file1",""  ! turn off all attributes
PROTECT "file2","S" ! System, but not R or H
```

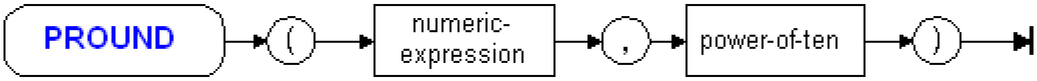
See Also:

CAT, CHECKREAD, COPY, CREATE, INITIALIZE, MSI, PRINT LABEL PURGE, READ LABEL, RENAME, SYSTEM$("MSI")

PROUND

Rounds the argument to the specified power of ten.

Syntax: PROUND(numeric-expression, power-of-ten)



Sample: `Logic=PROUND(Express,-2)`

```
PRINT PROUND(Amount,Degree)
```

View Sample: **PROUND.BAS** **(also found in examples directory)**

Description:

The power-of-ten is a numeric expression, which is rounded to an integer. It specifies the digit position where the number should be rounded. Positive values are to the left of the decimal point and negative values are to the right. For example, **PROUND**(PI,0) rounds to the nearest integer (10^0) and **PROUND**(PI,-2) rounds to the nearest hundredth (10^(-2)).

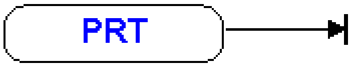
See Also:

CINT, DROUND, FIX, FRACT, INT, REAL

PRT

Returns the default device selector for the printer.

Syntax: PRT



Sample:

```
PRINTER IS PRT
PRINT "Default PRT is",PRT
```

View Sample: **PRT.BAS** **(also found in examples directory)**

Description:

The **PRT** function returns a constant representing the conventional printer interface select code. **PRT** exists to provide a useful mnemonic for the most common device selector for a printer. While **PRT** returns the conventional device selector for a printer, any legal device selector may be used in place of **PRT** in the PRINTER IS command (see PRINTER IS). The following are several common examples:

```
PRINTER IS 9     !serial printer
PRINTER IS CRT   !the display
PRINTER IS 70102 !2 IEEE-488 printers
```

PRT returns the constant 10. This is different from HP BASIC, which returns the constant 701. On the PC, most printers are connected to the parallel printer port, making 10 the most common printer device selector. With HP BASIC, most printers are connected to the HP-IB interface and have a primary address of 1, making 701 the most common printer device selector. To provide compatibility with existing software, the HTBasic **PRT** can be redefined to 701 (or any other value) with the CONFIGURE PRT statement.

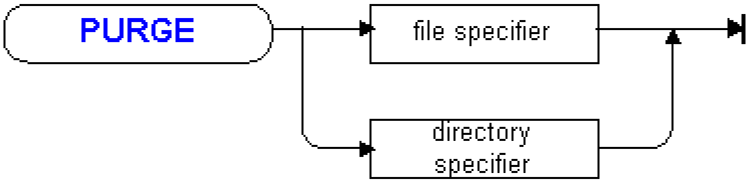
See Also:

CONFIGURE PRT, CRT, KBD, PRINTER IS

PURGE

Deletes a file or a directory on a mass storage media.

Syntax: PURGE { file-specifier | directory-specifier }



Sample: `PURGE "Work"`
`PURGE "ADir/BDir/Cdir"`

View Sample: **PURGE.BAS** **(also found in examples directory)**

Description:

The **PURGE** statement is used to delete a file or a directory. All data in the file is lost when the file is purged. **PURGE** will not delete a directory unless there are no files in that directory (except "." and ".."). The directory can not be the root directory and it can not be the current directory.

Neither a file nor a directory can be deleted if it has the read-only attribute. Use the PROTECT statement to clear the attribute before deleting the file. Windows does not allow an open file to be deleted. To delete a file or directory you must have the proper permissions.

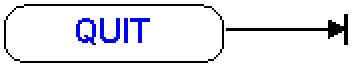
See Also:

CAT, COPY, CREATE, INITIALIZE, LINK, MASS STORAGE IS, PRINT LABEL , PROTECT, READ LABEL, RENAME, SYSTEM$("MSI")

QUIT

Closes the BASIC child window and returns to blank parent window.

Syntax: QUIT



Sample: `QUIT`

View Sample: **QUIT.BAS** **(also found in examples directory)**

Description:

QUIT is used to close the open program and return to a blank parent window. To close the entire HTBasic application use QUIT ALL.

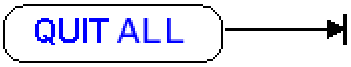
See Also:

EXECUTE

QUIT ALL

Quits BASIC and returns to the operating system.

Syntax: QUIT ALL



Sample: `QUIT ALL`

View Sample: **QUITALL.BAS** **(also found in examples directory)**

Description:

QUIT ALL is used to leave the BASIC programming environment and return to the computer's operating system. If the program is in a paused state, a STOP is automatically executed to close any open files before quitting.

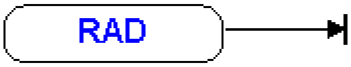
See Also:

EXECUTE

RAD

Sets the trigonometric mode to radians.

Syntax: RAD



Sample: `RAD`

View Sample: **RAD.BAS** **(also found in examples directory)**

Description:

All angle arguments and functions that return an angle measurement use the current trigonometric mode which can be either radians or degrees. **RAD** sets the trigonometric mode to radians. The default trigonometric mode at start-up or after a SCRATCH A is radians. A subprogram will use the same trigonometric mode as its caller unless it executes a **RAD** or DEG statement. Upon returning to the caller the previous trigonometric mode is restored.

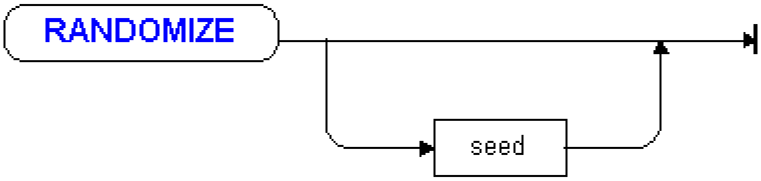
See Also:

ACS, ASN, ATN, COS, DEG, SIN, TAN

RANDOMIZE

Selects a seed for the RND function.

Syntax: RANDOMIZE [seed]



Sample: `RANDOMIZE`

```
RANDOMIZE Seed*PI
```

View Sample: **RANDOMIZE.BAS** **(also found in examples directory)**

Description:

The random number generator starting point is set to the user specified value. If no value is specified, the starting point is chosen at random. The seed value is a numeric expression rounded to an integer. If it is less than one, a value of one is used. If it is less than 2^31-2, its value is used. If it is larger, then 2^31-2 is used. The seed is reset to 37,480,660 at start-up, SCRATCH A, SCRATCH, and program prerun.

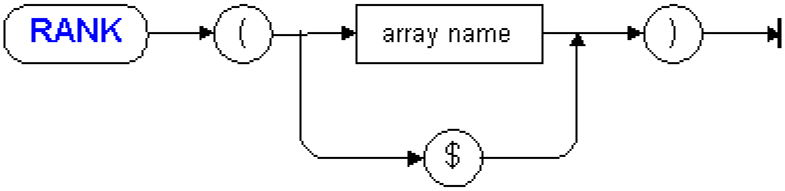
See Also:

RND

RANK

Returns the number of dimensions in an array.

Syntax: RANK(array-name[$])



Sample: `RANK(Color)`

```
RANK(File$)
IF RANK(A)=2 THEN PRINT "Two Dims"
```

View Sample: **RANK.BAS** **(also found in examples directory)**

Description:

RANK returns an INTEGER value from one to six that specifies the number of dimensions that are defined for the array.

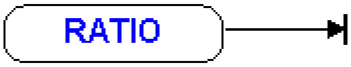
See Also:

BASE, DIM, MAXLEN, SIZE

RATIO

Returns the ratio of X to Y hard-clip limits for the PLOTTER IS device.

Syntax: RATIO



Sample:

```
WINDOW 0,RATIO,-1,1
Xmax=100*MAX(1,RATIO)
Ymax=100*MAX(1,1/RATIO)
```

View Sample: **RATIO.BAS** **(also found in examples directory)**

Description:

RATIO is useful for VIEWPORT and WINDOW calculations and for knowing the shape of the graphic screen or plotter paper.

See Also:

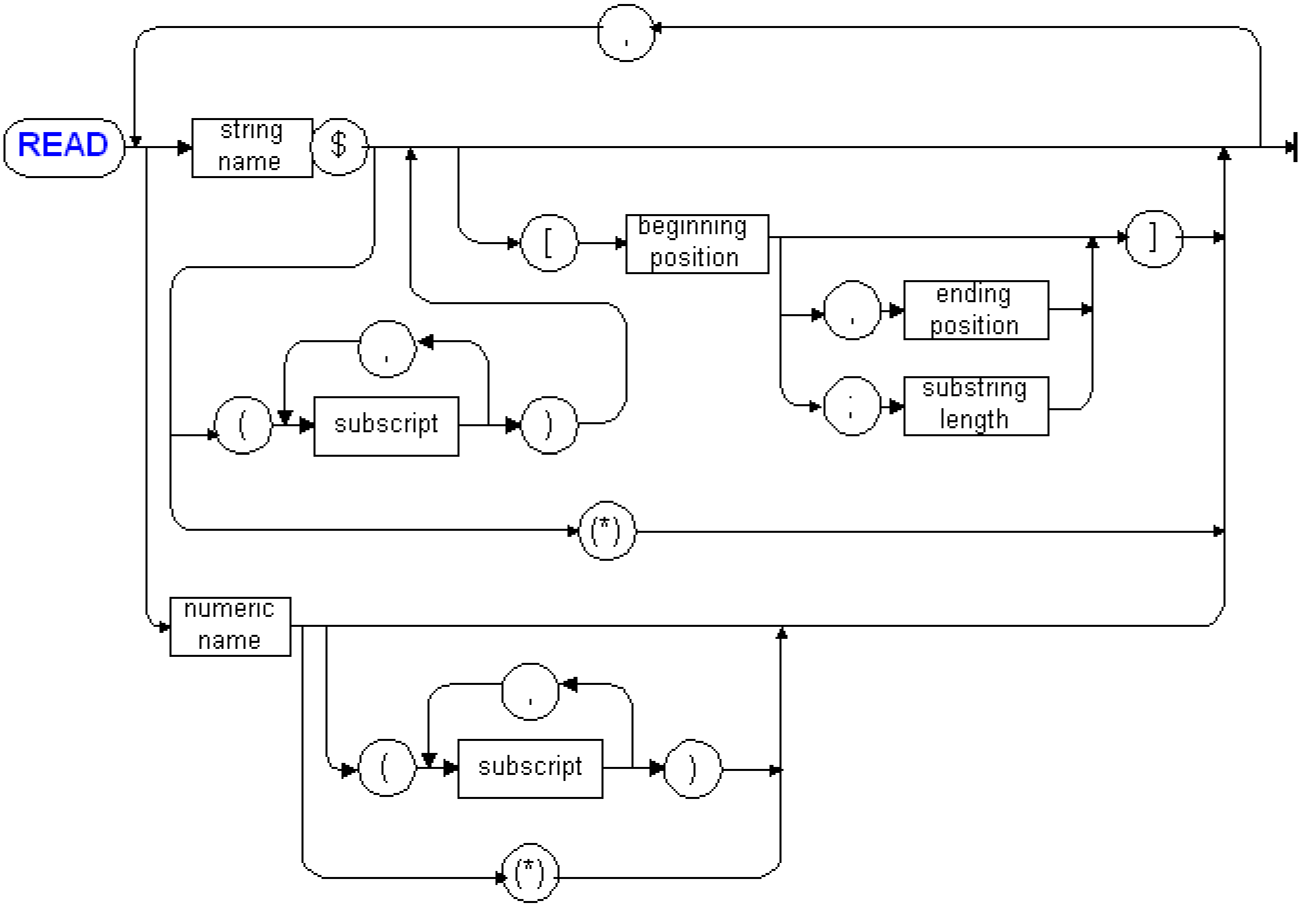
CLIP, SHOW, VIEWPORT, WINDOW

READ

Reads values from DATA statements.

Syntax: READ variable [,variable ...]

where: variable = variable-name[$] [(*)] |
numeric-name [(subscripts)] |
string-name$ [(subscripts)] [sub-string]
subscripts = subscript [,subscript...]



Sample:

```
READ Line,A$
READ Answer$(N)[20;5]
READ A,B,C(I,J)
READ Array(*)
```

View Sample: **READ.BAS** **(also found in examples directory)**

Description:

READ and DATA statements can conveniently initialize multiple variables from data embedded in the program. An array may be read in row-major order using the full array specifier, "(*)". DATA statements are stored as strings and the VAL function is used to read numeric values. The value is rounded to an integer if an integer variable is specified.

The first **READ** statement in a context reads the first DATA statement in that context. Each **READ** statement thereafter maintains a DATA pointer that moves to the next item after each is read from the DATA statement. The DATA pointer can be reset to the beginning of any DATA statement in the context with the RESTORE statement.

Complex numbers are read in rectangular form, the real part first, followed by the imaginary part. The two parts should be separated by a comma.

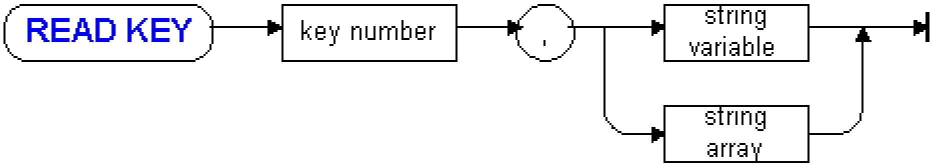
See Also:

DATA, RESTORE

READ KEY

Returns one or more softkey macro definitions.

Syntax: READ KEY key-number, string-variable$ [(subscripts)] [sub-string]
READ KEY key-number, string-array$(*)



Sample: `READ KEY 2,Keytwo$`

```
READ KEY First_key,Several_keys$(*)
```

View Sample: **READ KEY.BAS** **(also found in examples directory)**

Description:

Softkey macros defined with EDIT, LOAD or SET KEY can be read with this statement. The key-number is a numeric expression which is rounded to an integer and should be in the range zero through twenty-three. If a simple string or array element is specified, then only one key is returned. If a string array is specified, then successive keys, starting with the key-number specified, are returned into the elements of the string array.

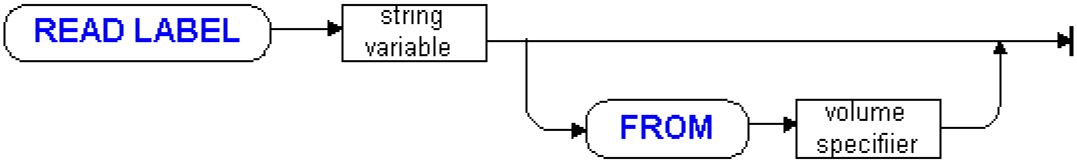
See Also:

EDIT KEY, LIST KEY, LOAD KEY, RE-STORE KEY, SCRATCH, SET KEY, STORE KEY

READ LABEL

Reads a volume label.

Syntax: READ LABEL string-variable$ [FROM volume-specifier]



Sample: `READ LABEL Id$`
`READ LABEL Name$ FROM Vol$`

View Sample: **READ LABEL.BAS** **(also found in examples directory)**

Description:

The volume label on the specified media is read and returned into the string variable. If no mass storage unit specifier is given, the MSI device is used.

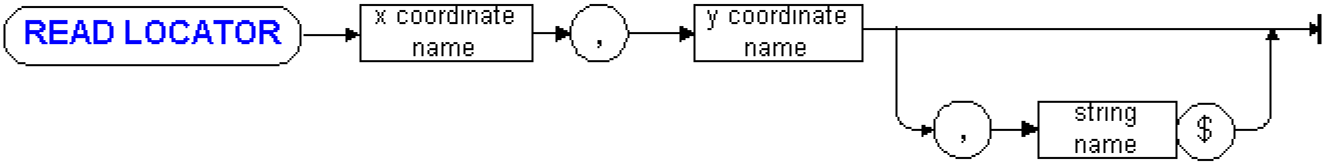
See Also:

PRINT LABEL

READ LOCATOR

Reads the locator device without waiting for a digitize operation.

Syntax: READ LOCATOR x-variable,y-variable [,string-name$]



Sample: `READ LOCATOR X,Y`

```
READ LOCATOR Xcoor,Ycoor,Position$
```

View Sample: **READ LOCATOR.BAS** **(also found in examples directory)**

Description:

The locator device position is read into the X and Y variables without waiting for a digitize operation. The current GRAPHICS INPUT IS device coordinates are in default units or the units defined in a WINDOW or SHOW statement. The optional string variable will receive the 8 byte status message defined as follows:

Byte	**Meaning**
1	Button Status - Status of the digitizing button on the locator. If the character is a "1", then the button is pressed; if it is a "0", then the button is not pressed.
2	Comma delimiter character.
3	Clip Indicator - If the character is a "0", then the point is outside the hard-clip limits. If a "1", the point is inside the hard-clip limits, but outside the soft-clip limits (clipping rectangle - see CLIP). If a "2" then it's inside the soft-clip limits.
4	Comma delimiter character.
5	Tracking ON/OFF - If the character is a "0", then tracking is off; if a "1", then tracking is on.
6	Comma delimiter character.
7-8	Button Positions - If S$ is the status string and B is the button number you wish to test, then BIT(VAL(S$[7,8]), B-1) returns one if B is down and zero if B is up.

See Also:

DIGITIZE, GRAPHICS INPUT IS, SET ECHO, SET LOCATOR, TRACK, WHERE

READIO

Reads a hardware register or a memory byte/word.

Syntax:

READIO(interface-select-code, hardware-register)
READIO(special-interface, address)
READIO(9827, simple-var)

where:

hardware-register = numeric-expression rounded to an integer
special-interface = numeric-expression rounded to an integer, legal values are explained in the text
address = numeric-expression rounded to a linear address
simple-var = numeric-name | numeric-array-element

Sample:

```
Control=READIO(Centronix,2)
Shift_flag=READIO(9826,&H417)
```

View Sample: **READIO.BAS (also found in examples directory)**

Description:

Hardware Registers

The contents of a hardware interface register are read and returned on the specified interface. Do not mix **READIO** WRITEIO operations with STATUS/CONTROL operations. Do not attempt to use **READIO** WRITEIO registers unless you are very familiar with the hardware; use the STATUS/CONTROL registers instead. Consult the hardware manuals for your computer for complete documentation on interface hardware. The *User's Guide* lists the **READIO** WRITEIO registers for the interface device drivers included with HTBasic. For other device drivers, the documentation included with the driver lists the register definitions.

READIO WRITEIO registers in HTBasic are not compatible with HP BASIC **READIO** WRITEIO registers when the interface hardware is not the same. TransEra's IEEE-488 and HP's HP-IB use the same IEEE-488 chip; therefore, the **READIO** WRITEIO registers are identical. The serial interface hardware registers differ not only if the UART chip is different, but also if the circuitry surrounding the chip is different. The TransEra GPIO is designed to be **READIO** WRITEIO compatible with HP's GPIO.

Special Interface Select codes

There are a number of special interface select codes which can be read with the **READIO** statement. The legal values for special-interface are given in the following paragraphs. For compatibility with earlier releases of HTBasic, **READIO**(8080,L) and **READIO**(-8080,L) are still supported but have been replaced with INP and INPW, respectively.

PEEK Memory

READIO(9826,L) and **READIO**(-9826,L) are used to "peek" at the contents of a memory byte or word, respectively. L specifies the address of the byte/word to peek. If peeking a word and L is odd, the even address L-1 is used. Where L specifies an address within the HTBasic process.

Warning: Peek should only be done on addresses returned by **READIO**(9827,I)! Peeking any other location can cause your system to crash, data to be lost and damage to your computer hardware. Use of this function for any other address is unsupported, and TransEra cannot be held responsible for any consequences.

Locating a Numeric Variable

READIO(9827,I) is used to locate the variable I. **READIO**(9827,A(0)) is used to locate the address of the first element of A. These operations are useful when a small assembly subroutine is stored in a variable and called with WRITEIO.

See Also:

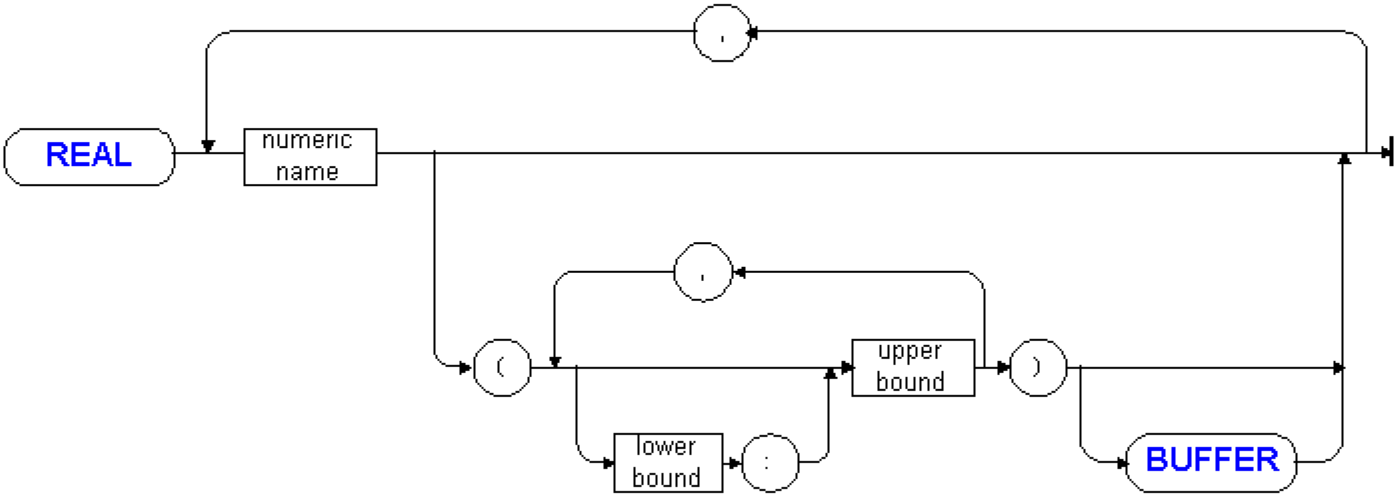
CONTROL, INP, OUT, STATUS, WRITEIO

REAL

Reserves storage for floating point variables and arrays.

Syntax: REAL variable [,variable...]

where: variable = numeric-name [(bounds) [BUFFER]]
bounds = [lower-bound :] upper-bound [,bounds]
lower and upper-bound = integer constant in the range -32768 to 32767



Sample:

```
REAL X,Buf(500) BUFFER
REAL Volts(-10:10,4)
```

View Sample: **REAL.BAS** **(also found in examples directory)**

Description:

REAL declares, dimensions and reserves memory for floating point variables and arrays. **REAL** variables use eight bytes of storage space. An array's maximum dimension is six and each dimension can hold a maximum of 32,767 elements. If a lower bound is not specified, the default is the OPTION BASE value (0 or 1). A **REAL** variable may be declared a buffer by specifying **BUFFER** after the variable name. **BUFFER** variables are used with the TRANSFER statement.

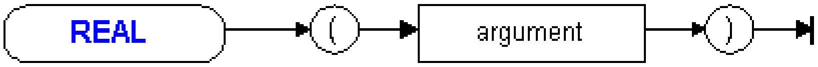
See Also:

ALLOCATE, DEF FN, COM, COMPLEX, DIM, INTEGER, SUB, TRANSFER

REAL

Converts an INTEGER or COMPLEX number to REAL.

Syntax: REAL(numeric-expression)



Sample:

```
PRINT REAL(Z)
DRAW REAL(C),IMAG(C)
```

View Sample: **REAL.BAS** **(also found in examples directory)**

Description:

The real part of a complex number is returned with **REAL**, and the imaginary part with IMAG. To express the parts of a complex number in polar form, use ABS and ARG:

```
PRINT "Rectangular form: Real = ";REAL(Z),"Imag =";IMAG(Z)
PRINT "Polar form: Magnitude = ";ABS(Z),"Angle = ";ARG(Z)
```

See Also:

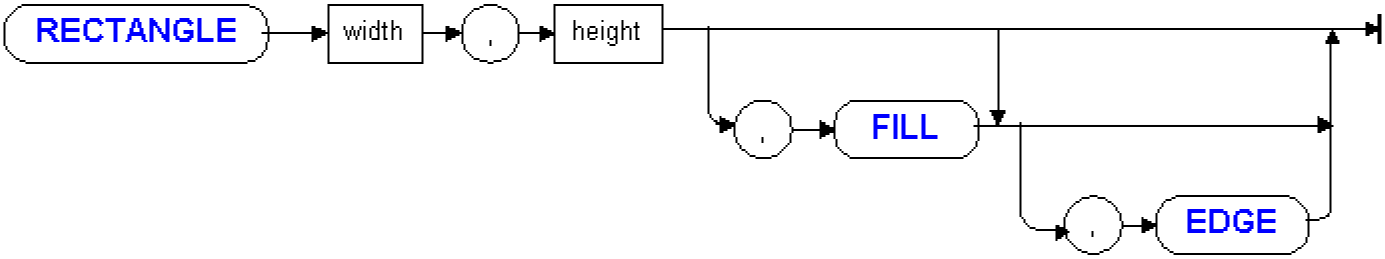
ABS, ARG, CMPLX, CONJG, IMAG

RECTANGLE

Draws and optionally fills and edges rectangles.

Syntax: RECTANGLE width,height [,FILL] [,EDGE]

where: width and height = numeric-expressions



Sample: `RECTANGLE 10,25`
`RECTANGLE 8,-10,FILL,EDGE`

View Sample: **RECTANGLE.BAS** **(also found in examples directory)**

Description:

A rectangle is a polygon described by its width and height displacement from the current pen position.

The signs of the width and height determine the position of the rectangle relative to the current pen position. If the width is positive, the pen position is on a left corner of the rectangle. If the width is negative, the pen position is on a right corner of the rectangle. If the height is positive, the pen position is on a lower corner of the rectangle. And if the height is negative, the pen position is on an upper corner.

The rectangle can be filled with the current AREA color and edged with the current PEN color and LINE TYPE. If neither are specified, **EDGE** is assumed.

The PIVOT and PDIR statements affect the **RECTANGLE** statement.

See Also:

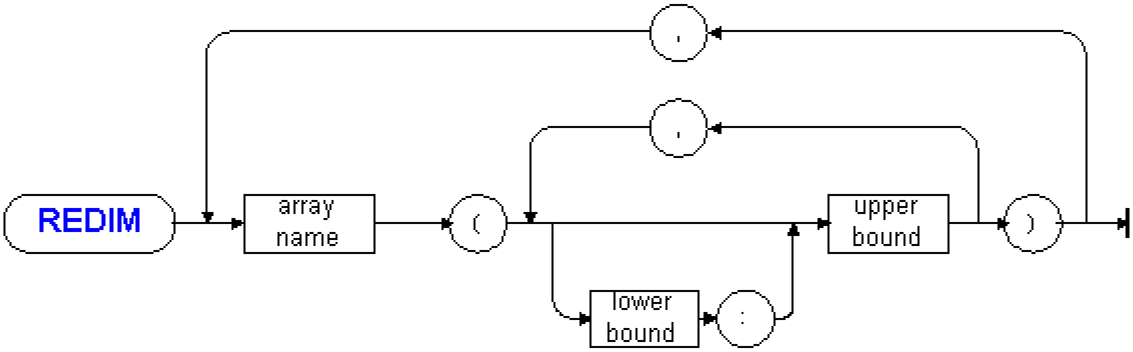
LINE TYPE, PDIR, PEN, PIVOT, PLOT, POLYGON, POLYLINE

REDIM

Redimensions an array by changing the subscript ranges.

Syntax: REDIM array-name[$](bounds) [,array-name[$](bounds)...]

where: bounds = [lower-bound:] upper-bound [,bounds]
lower and upper-bound = numeric-expressions rounded to integers



Sample: `REDIM Array(Lowbnd:Upbnd)`
`REDIM Myarray$(I,J,K,L)`

View Sample: **REDIM.BAS** **(also found in examples directory)**

Description:

An array can only be redimensioned if the number of dimensions is the same as in the original DIM statement and the total number of elements does not exceed the total in the DIM statement. Also, to redimension an array declared in a COM statement, the COM declaration must include subscript information (as opposed to a full array specifier, "(*)").

REDIM does not change the values presently stored in memory, but because the number of elements in each dimension might change, the values in each element may appear to "move" to another element.

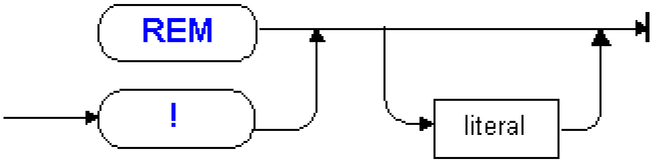
See Also:

ALLOCATE, COM, COMPLEX, DIM, INTEGER, REAL

REM

Begins a REMark or comment line for program documentation.

Syntax: REM *any text*
program statement ! *any text*



Sample:

```
REM This statement is not executed
Info=0  ! Clear flag byte
```

View Sample: **REM.BAS** **(also found in examples directory)**

Description:

A **REM** statement is used to insert comments into programs. The **REM** statement may contain any text you wish. It is useful in explaining what the program is doing. A comment tail, "!", is similar to the **REM** statement, however, the comment tail may appear on the same line as a program statement. Any text may appear to the right of the comment tail and is ignored when the line is executed. When an INDENT command is given, the position of a comment tail is left unchanged.

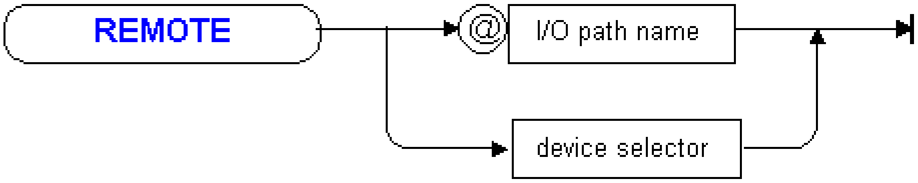
See Also:

EDIT, INDENT, REN

REMOTE

Sets the remote state on a IEEE-488 device.

Syntax: REMOTE {@io-path | device-selector}



Sample: `REMOTE Dev`
`REMOTE @Pwsply`

Description:

The IEEE-488 bus remote line is asserted. If the computer is the active controller and primary addresses are specified, it listen addresses the devices to switch them to remote mode. The remote line is asserted if the computer is the system controller and ISC select code is specified. The IO-path or device-selector must refer to one or more IEEE-488 devices or to the IEEE-488 interface select code.

If the computer is not the system controller or it is not the active controller and primary addresses are specified, an error is generated.

See Also:

ABORT , CLEAR, LOCAL, PASS CONTROL, PPOLL, REQUEST, SEND, SPOLL, TRIGGER

REN

Renumbers program lines.

Syntax: REN [start-number [,increment]] [IN begin-line [,end-line]]

where: line = line-number | line-label
increment = integer constant

Sample:

```
REN 1000 IN 100,800
REN 1200
REN 100,5
REN 150,1 IN 140,Mark
```

Description:

This statement renumbers program statements, including the line references in all program statements such as GOSUB and GOTO to coincide with the new line numbers.

You can optionally specify the starting position, the increment between lines or a range of lines to renumber. The default value for both the start line number and the increment is ten.

Note: You cannot specify a new starting line number that would cause the lines to change position with respect to other existing program lines. Use MOVELINES or COPYLINES to do this.

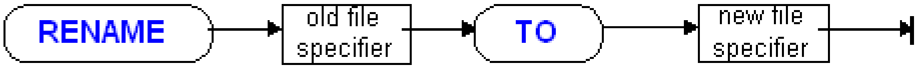
See Also:

COPYLINES, MOVELINES

RENAME

Changes the name of a file.

Syntax: RENAME old-file-specifier TO new-file-specifier



Sample:

```
XT=RENAME "PROG.DAT" TO "CURVE.DAT"
RENAME "X" TO "Xcalc"
RENAME Volume$&Old$ TO New$
```

View Sample: **RENAME.BAS** **(also found in examples directory)**

Description:

RENAME changes the name of a file. Both the old and the new names may be specified as string expressions. The new name must not already exist on the mass storage device.

If you are using **RENAME** to move a file from one place in a hierarchical file system to another, the HTBasic **RENAME** requires that both file specifiers be complete and both directories be on the same mass storage device. HTBasic does not require that the destination be on the same mass storage device.

See Also:

CAT, COPY, CREATE, INITIALIZE, MASS STORAGE IS, PRINT LABEL, PROTECT, PURGE, READ LABEL, RENAME, SYSTEM$("MSI")

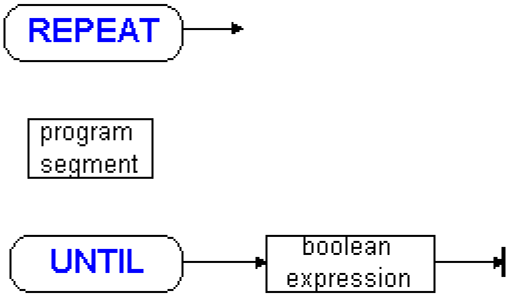
REPEAT ... UNTIL

Defines a loop that is repeated UNTIL a condition is satisfied.

Syntax:

REPEAT
statements
UNTIL numeric-expression

where: statements = zero, one or more program statements



Sample:

```
770 REPEAT
780   CALL Test(X)
790   X=X+Next
800 UNTIL X=Last
```

View Sample: **REPEAT UNTIL.BAS** **(also found in examples directory)**

Description:

The statements between the **REPEAT** and **UNTIL** are first executed. When the **UNTIL** statement is reached, the expression is evaluated. If the expression is false (zero), the statements between the **REPEAT** and **UNTIL** are executed again. If the expression is true (non-zero), execution continues with the statement following the **UNTIL**.

See Also:

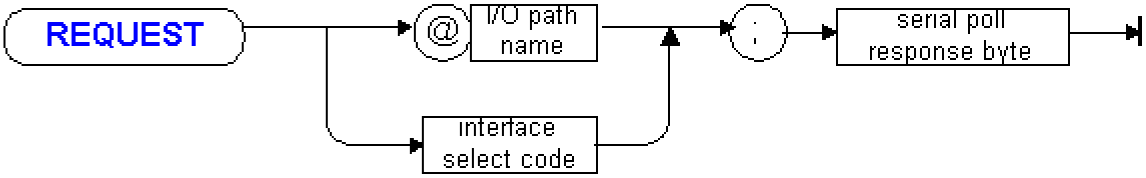
FOR, LOOP, SELECT, WHILE

REQUEST

Sends a Service Request SRQ on the IEEE-488.

Syntax: REQUEST {@io-path | interface-select-code} ; response-value

where: io-path = I/O path assigned to the IEEE-488 interface
response-value = numeric-expression rounded to an integer



Sample:

```
REQUEST @Gpib;Serialpoll
REQUEST Isc;BINIOR(Bit3,64)
```

Description:

A Service Request, SRQ, is sent by a non-active controller on the IEEE-488 bus. If the computer is the active controller or if the device-selector or the IO-path specifies address information, an error is generated.

To request service, the response value must have bit six set. The SRQ line will remain set until polled by the active controller or another **REQUEST** statement is executed with bit six clear.

See Also:

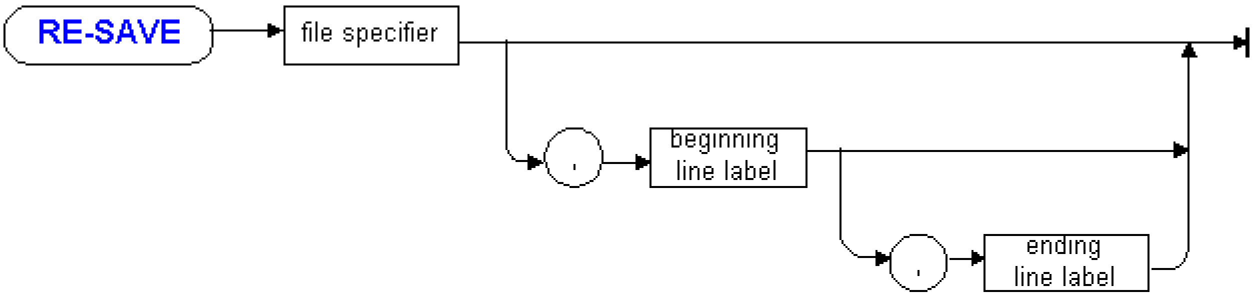
ABORT , CLEAR, LOCAL, PASS CONTROL, PPOLL, REMOTE, SEND, SPOLL, TRIGGER

RE-SAVE

Copies the program into the specified ASCII file.

Syntax: RE-SAVE file-specifier [,start-line [,end-line]]

where: line = line-number | line-label



Sample:

```
RE-SAVE "Story"
RE-SAVE "CALPROG",1000,2000
RE-SAVE "TREE\BRANCH\FILE",Label1
```

Description:

RE-SAVE outputs any range of program lines to an ASCII file. The resulting program can be re-entered with the GET statement.

If the specified file already exists, the old contents are discarded before the SAVE takes place. The program is then stored out in the same format, ASCII (LIF ASCII) or ordinary (DOS ASCII, UNIX ASCII, etc.), as the previous file. If it does not exist, a new file is created whose type depends on the setting of CONFIGURE SAVE ASCII.

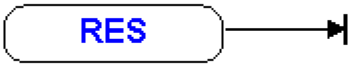
See Also:

CONFIGURE SAVE ASCII, GET, LIST, LOAD, RE-STORE, SAVE, STORE

RES

Returns the result of the last numeric keyboard calculation.

Syntax: RES



Sample:

```
Sum=RES+Sum
PRINT "User Response:";RES
```

View Sample: RES.BAS **(also found in examples directory)**

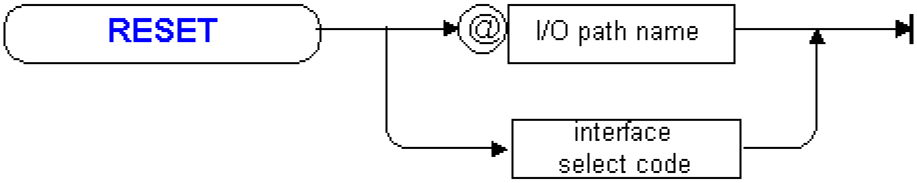
Description:

Typing in a numeric or string expression and pressing ENTER causes the computer to evaluation the expression and print the result on the message line. This is called "calculator mode" and allows you to use your computer as you would a hand calculator. If the result is numeric, it is saved for later recall by using the **RES** function.

RESET

Resets an interface or file or buffer pointers.

Syntax: RESET {@io-path | interface-select-code}



Sample: `RESET 9`

```
RESET Gpib
RESET @Buff
```

View Sample: RESET.BAS **(also found in examples directory)**

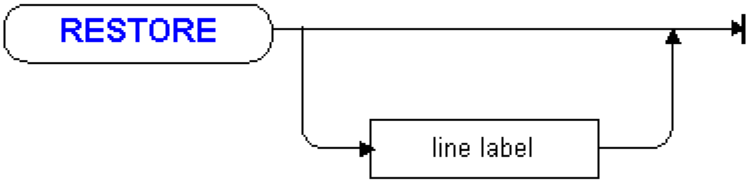
Description:

The **RESET** statement directed to an interface performs an interface reset. When directed to a file it sets the file position pointer to the beginning of the file. When directed to a buffer it sets all buffer control entries to their initial values with the empty and fill pointers set to one and all other entries set to zero.

RESTORE

Specifies which DATA statement to use for the next READ operation.

Syntax: RESTORE [{line-number | line-label}]



Sample:

```
RESTORE
RESTORE 950
RESTORE Star
```

View Sample: **RESTORE.BAS** **(also found in examples directory)**

Description:

The next READ statement gets its data from the current data pointer. **RESTORE** sets the data pointer to the specified program line. If that line is not a DATA statement the next higher numbered DATA statement will be used for the next READ statement. If no line is specified, the data pointer is set to the first DATA statement in the current context.

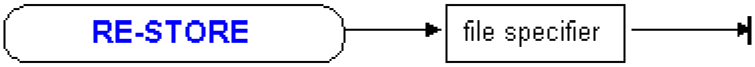
See Also:

DATA, READ

RE-STORE

Stores the BASIC program in a file.

Syntax: RE-STORE file-specifier



Sample:
```
RE-STORE "FFT"
RE-STORE Volume$&Myfile$
```

View Sample: **RE-STORE.BAS** **(also found in examples directory)**

Description:

The program currently in memory is STOREd in the file in binary form. If the file already exists, it must be a PROG file. The old contents are discarded and the file is replaced with the current program in memory. If it does not exist, a new PROG file is created.

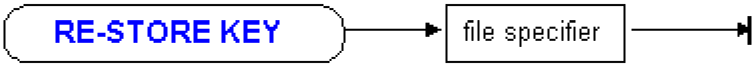
See Also:

GET, LIST, LOAD, RE-SAVE, RE-STORE KEY, SAVE, STORE

RE-STORE KEY

Stores the KEY definitions in a file.

Syntax: RE-STORE KEY file-specifier



Sample:

```
RE-STORE KEY "Definition"
RE-STORE KEY "A:KEYS"
```

View Sample: **RE-STORE KEY.BAS** **(also found in examples directory)**

Description:

Softkey macro definitions are stored into the specified file. If the file already exists, the old contents are discarded and the present key definitions are stored. If it does not exist, a new BDAT file is created.

Using FORMAT OFF, the definition for each defined softkey is written to the file by outputting two items. The first item is an integer, specifying the key number. The second item is a string, giving the key definition.

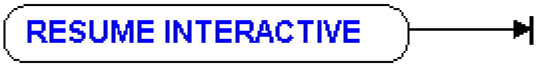
See Also:

EDIT KEY, LIST KEY, LOAD KEY, READ KEY, SCRATCH, SET KEY, STORE KEY

RESUME INTERACTIVE

Restores the normal functions of program control keys.

Syntax: RESUME INTERACTIVE



Sample: `RESUME INTERACTIVE`

View Sample: **RESUME INTERACTIVE.BAS** **(also found in examples directory)**

Description:

The normal functions of the program control keys CLR I/O, ENTER, PAUSE, RESET, STEP and STOP are enabled. These keys are disabled by SUSPEND INTERACTIVE.

See Also:

SUSPEND INTERACTIVE

RETURN

Returns to the program line following the last GOSUB line.

Syntax: RETURN

Sample:

```
200 GOSUB 300
...
299 STOP
300 PRINT A,B,C
310 RETURN
```

View Sample: **RETURN.BAS** **(also found in examples directory)**

Description:

The GOSUB statement transfers control to a subroutine; the **RETURN** statement transfers control back to the next statement following the GOSUB. You can have many GOSUBs to the same subroutine and a **RETURN** occurring in that subroutine returns control to the statement following the specific GOSUB used to get to the subroutine. You can only enter a subroutine by using GOSUB. If you don't use GOSUB, the **RETURN** statement causes an error when executed.

The **RETURN** keyword is also used to return values from user-defined functions. See DEF FN for an explanation of **RETURN** used in this way.

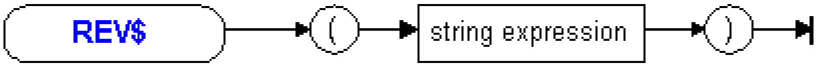
See Also:

DEF FN, GOSUB

REV$

Reverses the sequence of characters in a string.

Syntax: REV$(string-expression)



Sample:

```
Backward$=REV$(Forward$)
Print REV$("radaR")
```

View Sample: REV$.BAS **(also found in examples directory)**

Description:

A string that contains the reverse sequence of characters of its argument is returned. This can help when searching for the last occurrence of a string.

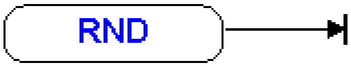
See Also:

CHR$, LWC$, NUM, RPT$, POS, TRIM$, UPC$, VAL, VAL$

RND

Returns a pseudo-random number.

Syntax: RND



Sample:

```
IF RND>0.25 THEN GOTO Start
Percent=RND*100
```

View Sample: **RND.BAS** **(also found in examples directory)**

Description:

A pseudo-random number greater-than zero and less-than one is returned. A seed value determines the starting point in the series. The seed can be modified using the RANDOMIZE statement. The default seed at start-up, SCRATCH, SCRATCH A and prerun is 37,480,660. The series of numbers returned is not guaranteed to be the same on different versions of HTBasic

See Also:

RANDOMIZE

ROTATE

Shifts a 16 bit binary value with wraparound.

Syntax: ROTATE(numeric-expression, distance)

where: distance = numeric-expression rounded to an integer

Sample:

```
B1=ROTATE(B2,5)
Word=ROTATE(Word,Places)
```

View Sample: ROTATE.BAS **(also found in examples directory)**

Description:

The numeric expression is rounded to an integer. The resulting integer, in binary form, is rotated the specified distance. The distance must be in the range 0 to ±15. If the distance is positive, then bits are moved to the right. Any bits moved out of the right-most bit (the least significant bit) are moved into the left-most bit (the most significant bit). If the distance is negative, then bits are moved to the left. Any bits moved out of the left-most bit are moved into the right-most bit.

For **ROTATE**(100,5) the number 100 is treated as a binary number and is rotated right five bits as follows:

100	= 0000000001100100
ROTATE(100,5)	= 0010000000000011

The result is returned as the decimal integer, 8195.

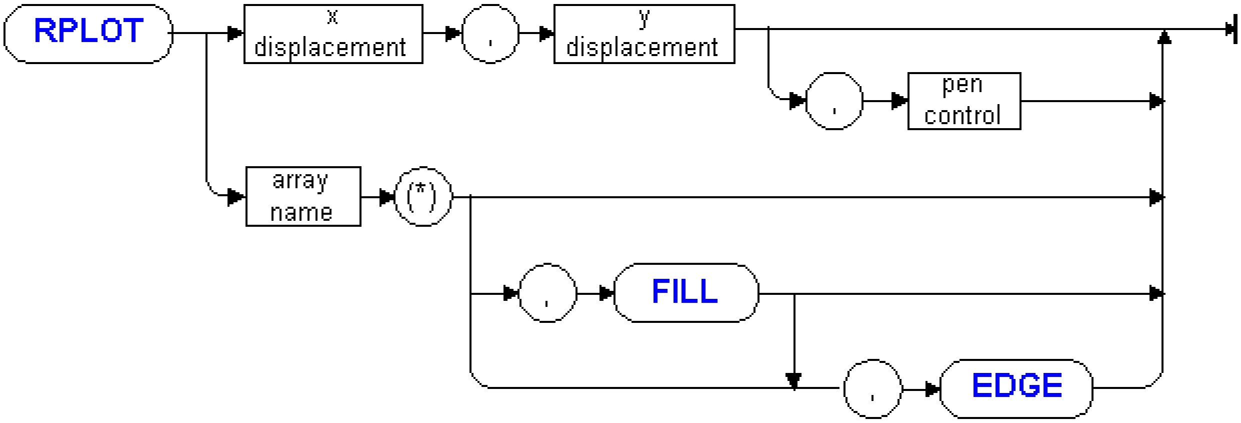
See Also:

BINAND, BINCMP, BINEOR, BINEQV, BINIMP, BINIOR, BIT, SHIFT

RPLOT

Moves the pen relative to the current graphic location.

Syntax: RPLOT x-displacement, y-displacement [,pen-control]
RPLOT numeric-array(*) [,FILL] [,EDGE]



Sample: `RPLOT 5,2`

```
RPLOT 5,-2,-1
RPLOT Array(*)
RPLOT Vector(*),FILL,EDGE
```

View Sample: **RPLOT.BAS** **(also found in examples directory)**

Description:

RPLOT is the same as IPLOT except that it moves the pen relative to the local origin. The local origin is the logical pen position after one of the following statements: AXES, DRAW, FRAME, GINIT, GRID, IDRAW, IMOVE, IPLOT, LABEL, MOVE, PLOT, POLYGON, POLYLINE, RECTANGLE and SYMBOL. See PLOT for a full explanation of **RPLOT** arguments.

The PIVOT and PDIR statements affect the **RPLOT** statement.

See Also:

AREA, CLIP, DRAW, IPLOT, MOVE, PLOT, POLYGON, POLYLINE

RPT$

Returns a string replicated a specified number of times.

Syntax: RPT$(string-expression, repeat-count)

Sample:

```
A$=RPT$("!",100)
PRINT RPT$("*",50)
PRINT RPT$(" ",(Centervalue/2)
```

View Sample: **RPT$.BAS** **(also found in examples directory)**

Description:

The repeat count is a numeric expression rounded to an integer value. If it is zero, a zero length string is returned. If it is negative or the resulting string would be greater than 32,767 characters, an error is generated.

See Also:

CHR$, LWC$, NUM, REV$, POS, TRIM$, UPC$, VAL, VAL$

RUN

Starts program execution.

Syntax: RUN [line-number | line-label]

Sample:

```
RUN
RUN 1000
RUN Next
```

Description:

RUN is executed in two parts, prerun initialization and program execution.

The prerun part reserves memory space for variables declared in DIM, REAL, INTEGER, COMPLEX and COM statements or implied in the program context. Numeric variables are set to zero and string variables are set to zero length strings. Prerun also checks for multi-line syntax errors such as illegal program structure, array references and mismatched COM statements. If prerun detects any errors, they are reported to the user and the program halts.

If prerun detects no errors, the MAIN program is run starting at the beginning or if a program line or label is specified, it starts execution at the specified line. The program line or label must be in the MAIN context. The program runs normally until it encounters a PAUSE, a STOP or END statement, an error or a TRACE PAUSE line.

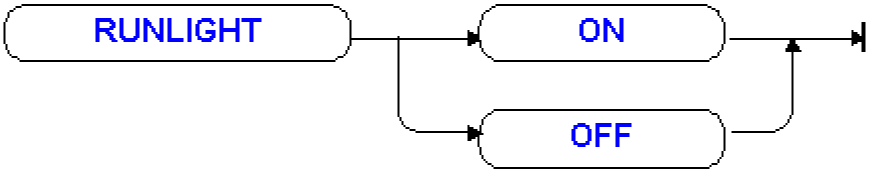
See Also:

CONT, END, LOAD, PAUSE, SCRATCH, STOP

RUNLIGHT

Controls the display of the pseudo runlight on the display.

Syntax: RUNLIGHT { ON | OFF }



Sample: `RUNLIGHT OFF`

View Sample: **RUNLIGHT.BAS** **(also found in examples directory)**

Description:

The pseudo **RUNLIGHT** is a single character in the lower right-hand corner of the display which indicates the state of HTBasic. By default, it is displayed. When doing screen dumps, the character can be unsightly so it is best to do a **RUNLIGHT OFF** before doing the dump. The meanings of the pseudo runlight characters are given in the following table.

Character	**Meaning**
?	Input
H	Help
*	Immediate command
R	Running
C	Change
F	Find
E	Edit
S	SUBs
-	Paused
(none)	Idle

See Also:

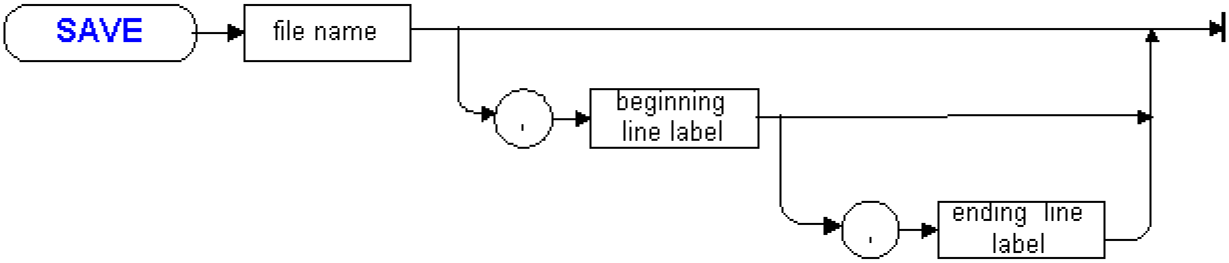
CLEAR LINE, CLEAR SCREEN, KEY LABELS

SAVE

Saves the current program into an ASCII file.

Syntax: SAVE file-specifier [,start-line [,end-line]]

where: line = line-number | line-label



Sample:

```
SAVE "DRAFTER"
SAVE "Pennies",100,Sort
SAVE "A:MYPROG"
```

View Sample: **SAVE.BAS** **(also found in examples directory)**

Description:

SAVE outputs any range of program lines to an ASCII file. Depending on the setting of CONFIGURE SAVE ASCII, the file type will either be ASCII (LIF ASCII) or ordinary (DOS ASCII, UNIX ASCII, etc.). The resulting program can be re-entered with the GET statement.

CONFIGURE SAVE ASCII sets the file type **SAVE** uses when saving a file to disk. The default setting, **ON**, produces a LIF ASCII file. This type of file is useful for exchanging programs with HP Workstations and for saving programs with string literals that contain embedded control characters such as carriage-returns or line-feeds in string literals since GET will interpret them as end-of-line indicators.

If the specified file already exists, **SAVE** generates an error message; whereas RE-SAVE will reuse an existing file.

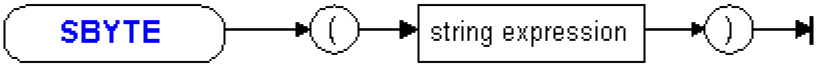
See Also:

GET, LIST, LOAD, RE-SAVE, RE-STORE, STORE

SBYTE

Checks for second byte of a two byte character.

Syntax: SBYTE(string)



Sample:

```
PRINT SBYTE(A$)
IF SBYTE(A$[I]) THEN PRINT "Two Bytes"
```

Description:

SBYTE is used with FBYTE to determine whether a character is one or two bytes long. SBYTE returns a one if the second byte of the string argument is in the valid range for the second byte of a two byte character.

This function is only available and enabled in specific versions of HTBasic.

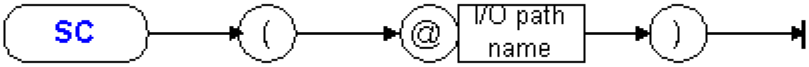
See Also:

CVT$, FBYTE

SC

Returns the interface select code associated with an I/O path name.

Syntax: SC(@io-path)



Sample: `Code=SC(@Dev)`

View Sample: **SC.BAS** **(also found in examples directory)**

Description:

Only the interface code is returned if the IO-path is assigned to a device-selector with primary addressing specified. A zero is returned if the IO-path name is assigned to a buffer.

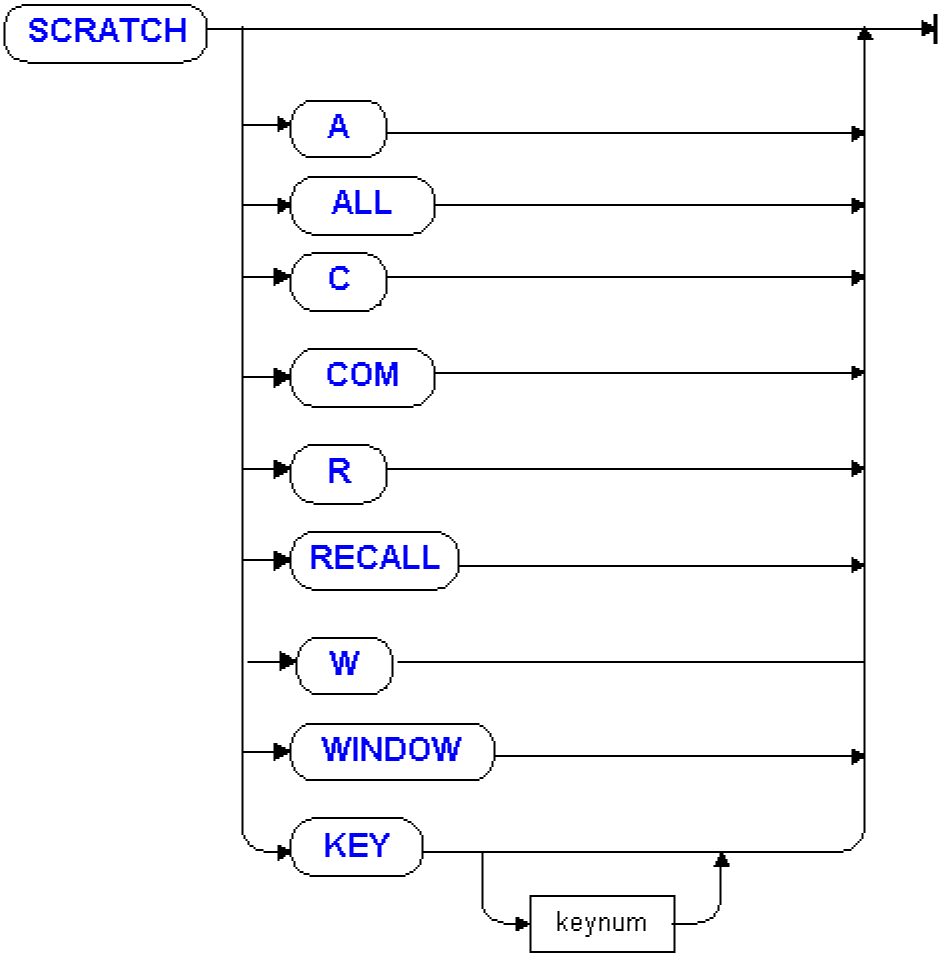
See Also:

ASSIGN

SCRATCH

Clears user memory.

Syntax: SCRATCH [A|ALL | B|BIN | C|COM | KEY [key-number] | R|RECALL]



Sample:

```
SCRATCH
SCRATCH KEY 2
SCRATCH C
```

View Sample: **SCRATCH.BAS** **(also found in examples directory)**

Description:

SCRATCH allows you to clear the BASIC program, program variables, COM variables, softkey macro definitions and the recall buffer. The following paragraphs explain each variation of **SCRATCH**:

SCRATCH

Deletes the current BASIC program and any variables not in COM.

SCRATCH A or ALL

SCRATCH A clears the BASIC program, all variables, including those in COM and all softkey macro definitions. Internal parameters are set to their default, start-up values. **SCRATCH ALL** is synonymous with **SCRATCH A**.

SCRATCH B or BIN

In HTBasic, **SCRATCH B** is equivalent to **SCRATCH A**. In HP BASIC, it deletes all **BIN**s except the

CRT driver in use. In HTBasic, **BIN**s are used for device drivers which can't be **SCRATCH**ed. **SCRATCH B** is synonymous with **SCRATCH BIN**.

SCRATCH C or COM

SCRATCH C clears all variables including those in COM, but leaves the BASIC program and the softkey macro definitions intact. **SCRATCH COM** is synonymous with **SCRATCH C**.

SCRATCH KEY [key-number]

Without the optional key number, this command clears all the softkey macro definitions. With the key number, only the specified key is cleared. The key-number may be a numeric expression which is rounded to an integer and must be in the range zero through twenty-three.

SCRATCH R or RECALL

SCRATCH R clears the keyboard **RECALL** buffer. **SCRATCH RECALL** is synonymous with **SCRATCH R**.

See Also:

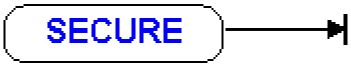
EDIT KEY, LIST KEY, LOAD KEY, READ KEY, RE-STORE KEY, SET KEY, STORE KEY

SECURE

Protects programs lines.

Syntax: SECURE [start-line-number [,end-line-number]]

where: line-number = integer constant



Sample:

```
SECURE
SECURE Payrolla,Payrollb
```

Description:

The **SECURE** command protects programs lines so they cannot be listed. Secured lines are listed as a line number followed by an asterisk "*" character. If no program lines are specified, all program lines are secured. If no end-line is specified, only the start-line is secured.

Warning: Once a line has been secured it can not be un-secured! Make sure that you have another copy of the program before you use the **SECURE** command.

See Also:

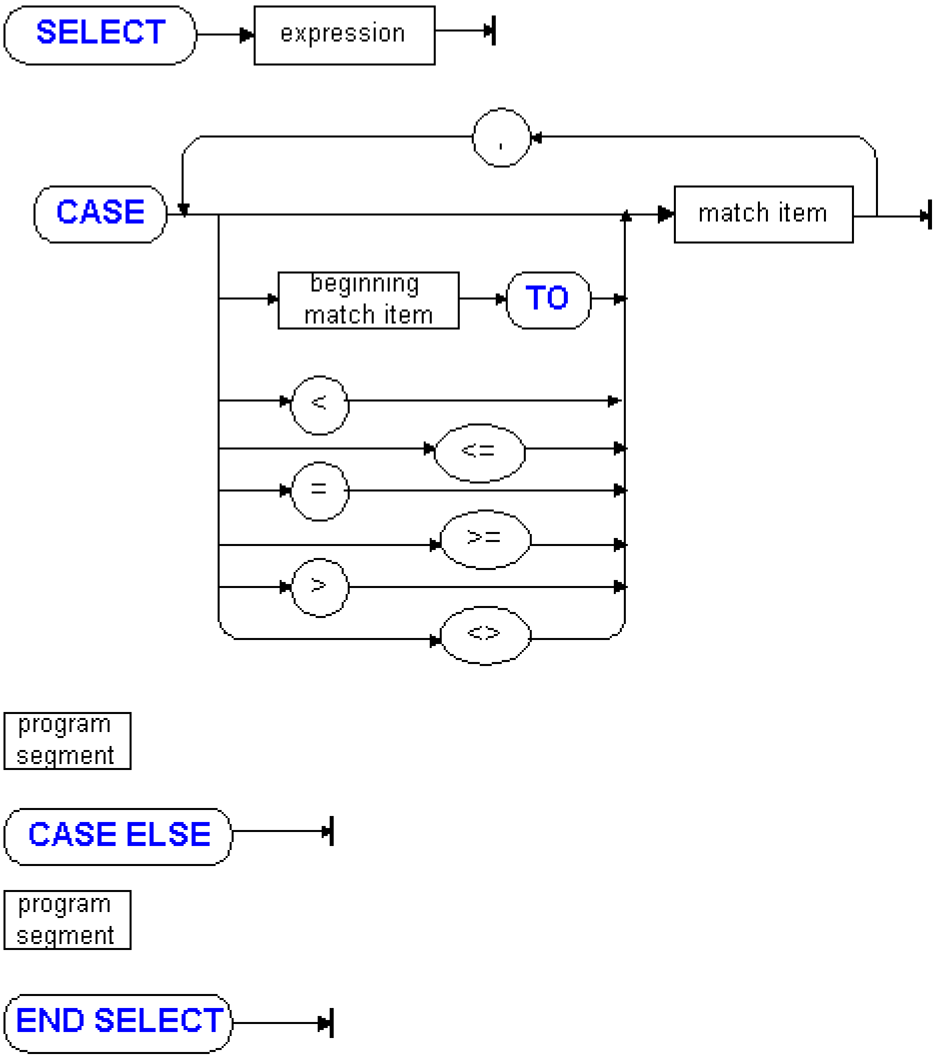
EDIT, LIST

SELECT ... CASE

Defines a CASE block structure.

Syntax: SELECT string-or-numeric-expression
CASE case-expression
statements
[CASE ELSE]
statements
END SELECT

where: statements = zero, one or more program statements
including additional CASE statements
case-expression = [relation] value [,case-expression]
relation = { < | <= | = | >= | > | <> | value TO }
value = string-or-numeric-expression



Sample:

```
10   SELECT Option$
20   CASE "B"
30     A=1
40   CASE "0" TO "9","y","n"
50     A=2
60   CASE ELSE
70     A=0
```

```
80  END SELECT
```

View Sample: SELECT CASE.BAS **(also found in examples directory)**

Description:

The **SELECT** and **END SELECT** statements enclose a **SELECT** structure. The **SELECT** statement specifies a numeric or string expression. Within the **SELECT** structure, **CASE** statements introduce alternative program sections to be executed based on the value of the **SELECT** statement expression. Each **CASE** statement type must match the type of expression in the **SELECT** statement. If a case-expression contains multiple values, the values are tested from left to right until a match is found. Any remaining expressions are not tested.

The **SELECT** expression value is used to test against each **CASE** statement value or range of values. The program statements following the first **CASE** statement to match are executed. Execution then continues at the line following the **END SELECT** statement. If none of the **CASE** statements match and there is an optional **CASE ELSE** statement, the program statements following the **CASE ELSE** will be executed, otherwise the entire **SELECT** structure is skipped.

While doing so is not encouraged, jumping into a **SELECT** structure with a GOTO is legal. Program statements are executed normally until a **CASE** statement is encountered. Execution then continues at the line following the **END SELECT** statement.

If there is an expression evaluation error in either the **SELECT** statement or one of the **CASE** statements the **SELECT** statement line number is reported with the error value.

Implementing ELSE IF

Although HTBasic does not have an explicit ELSE IF statement, it is possible to accomplish the same thing using a **SELECT** statement. Suppose you wish an ELSE IF construct like this:

```
10  IF X<-1 THEN
20    !do something here
30  ELSE IF Z=0 THEN
40    !do something else here
50  ELSE
60    !and something else here
70  END IF
```

This example can be accomplish using the **SELECT** statement as follows:

```
5   SELECT 1
10  CASE X<-1
20    !do something here
30  CASE Z=0
40    !do something else here
50  CASE ELSE
60    !and something else here
70  END SELECT
```

Line 5 states that the first case which evaluates to one will be executed. Since the result of a logical operator is 0 or 1, the first case with a logical expression that evaluates true will be executed.

See Also:

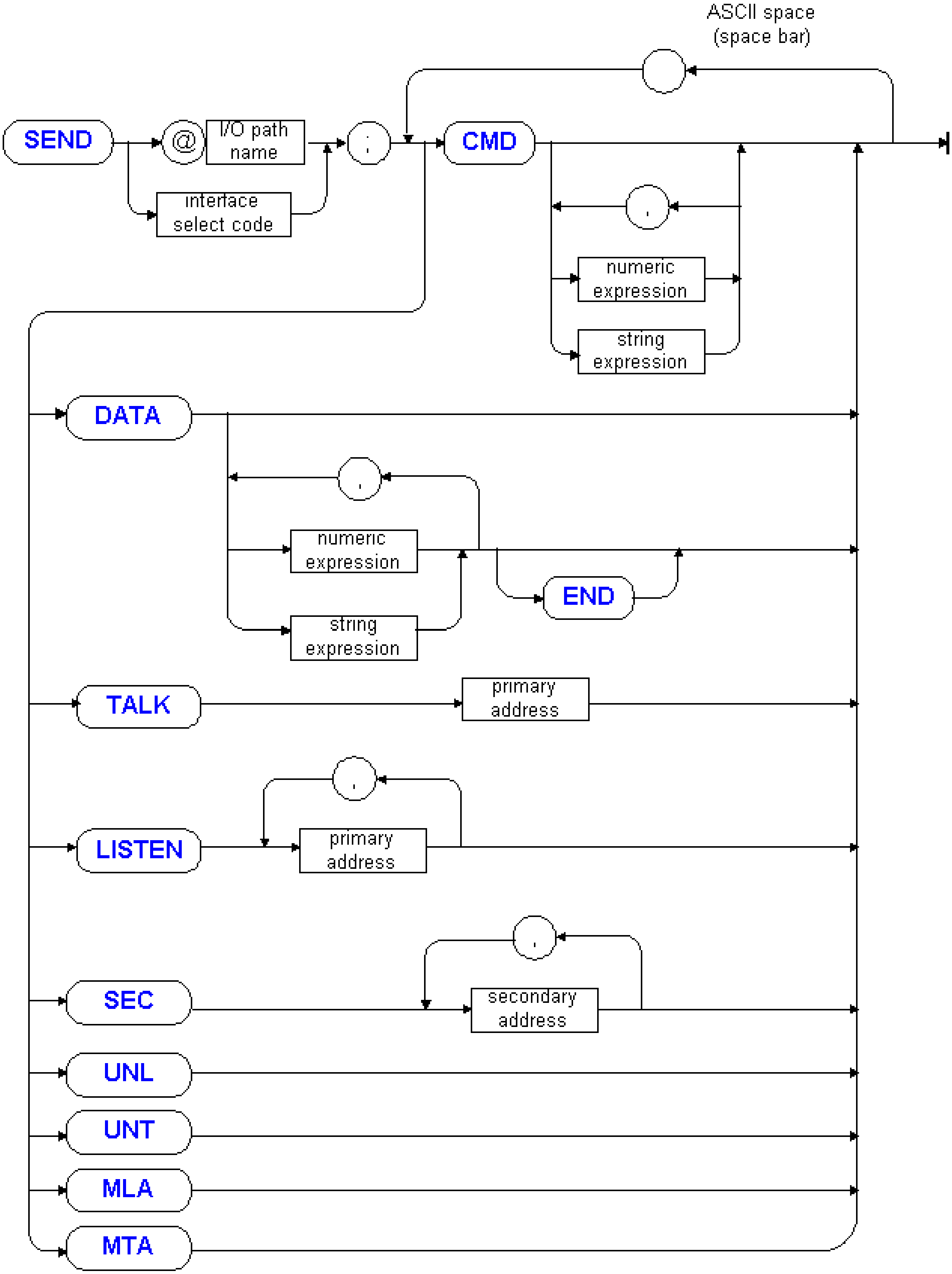
FOR, IF, LOOP, REPEAT, WHILE

SEND

Sends messages on the IEEE-488 bus.

Syntax: SEND dest ;message [message ...]

where: dest = {@io-path | interface-select-code}
io-path = I/O path assigned to the IEEE-488 interface
message = MTA | MLA | UNT | UNL |
CMD [expression-list] |
DATA [expression-list [END]] |
TALK primary-address |
LISTEN address-list |
SEC address-list
address-list = address [,address...]
address = numeric-expression rounded to an integer
expression-list = expression [,expression...]
expression = numeric-expression | string-expression


ASCII space
(space bar)
SEND
@
I/O path name
interface select code
;
CMD
,
numeric expression
string expression
DATA
,
numeric expression
string expression
END
TALK
primary address
LISTEN
,
primary address
SEC
,
secondary address
UNL
UNT
MLA
MTA

Sample:

```
SEND 7;UNL MTA LISTEN 2 DATA "Bye" END
SEND @Gpib;UNL MLA TALK Primary CMD 24+128
```

Description:

The **SEND** statement sends low level IEEE-488 commands and data bytes. IEEE-488 commands are sent with the ATN line asserted; whereas data bytes are sent without the ATN line asserted. The computer must be the active controller to use **CMD**, **TALK**, **UNT**, **LISTEN**, **UNL**, **SEC**, **MTA** or **MLA**. Any talk addressed device may send **DATA**.

Message	**Action Taken**
CMD	Sends the expression values as command bytes. **CMD** with no items asserts the ATN line.
DATA	Sends the expression values as data bytes. If **END** is added, EOI is set on the last data byte.
LISTEN	Sends the expression values as listen address commands.
MLA	Sends the Interface's Listen Address command.
MTA	Sends the Interface's Talk Address command.
SEC	Sends the expression values as secondary address commands.
TALK	Sends the expression value as a talk address command.
UNL	Sends the unlisten command.
UNT	Sends the untalk command.

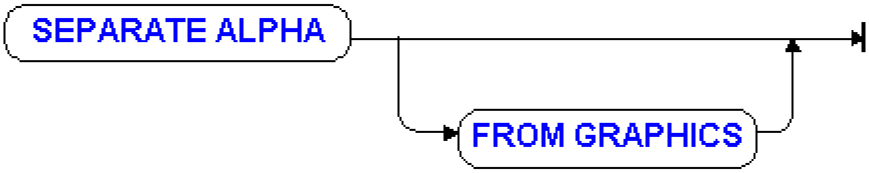
See Also:

ABORT, CLEAR, LOCAL, PASS CONTROL, PPOLL, REMOTE, REQUEST, SPOLL, TRIGGER

SEPARATE ALPHA FROM GRAPHICS

On a bit-mapped display, simulates 9836 style alpha/graphics hardware.

Syntax: SEPARATE ALPHA [FROM GRAPHICS]



Sample: `IF Display=8 THEN SEPARATE ALPHA FROM GRAPHICS`

View Sample: **SEPARATE ALPHA FROM GRAPHICS.BAS** **(also found in examples directory)**

Description:

SEPARATE ALPHA is the opposite of MERGE ALPHA WITH GRAPHICS. When separate, one or more bit plane is reserved for alpha text and the remaining planes are reserved for graphic output. The alpha and graphic planes can then be turned on or off or DUMPed independently. However, ALPHA text color and graphic pens are limited as shown in the table below.

The following table shows the colors available when **SEPARATE ALPHA FROM GRAPHICS** is used, depending on the total number of colors available.

Total Colors	Graph Pens	Black Alpha	White Alpha	Brown Alpha	Cyan Alpha
16	0-7	0	8	-	-
256	0-63	0	64	128	192

Porting Issues

HP BASIC assigns green to the first pen; HTBasic assigns white. If you prefer green or some other color, you must explicitly set a range of pen values to the color desired. The range starts with the white alpha pen value from the table above and continues to one less than the value of the brown alpha pen value. For 16 color systems, the last value should be 15. For example, the following code changes the alpha pen from white to green on a 16 color display:

```
10  SEPARATE ALPHA FROM GRAPHICS
20  PLOTTER IS CRT,"INTERNAL";COLOR MAP
30  FOR I=8 TO 15
40    SET PEN I INTENSITY 0,1,0
50  NEXT I
60  END
```

See Also:

MERGE ALPHA WITH GRAPHICS

SET ALPHA MASK

Determines which plane(s) can be modified by ALPHA display operations.

Syntax: SET ALPHA MASK numeric-expression

Sample:

```
SET ALPHA MASK Frame
SET ALPHA MASK 2
SET ALPHA MASK IVAL("1010",2)
IF Frame=5 THEN SET ALPHA MASK 3
```

Description:

The numeric expression value specifies which display bit planes are modified by alpha display operations. This statement does not affect monochrome displays. This statement is equivalent to CONTROL CRT,18.

This statement is not supported by HTBasic. Use MERGE ALPHA or SEPARATE ALPHA.

See Also:

ALPHA HEIGHT, ALPHA PEN, CLEAR SCREEN, MERGE ALPHA, SEPARATE ALPHA, SET DISPLAY MASK

SET CHR

Defines the bit-patterns for one or more characters.

Syntax: SET CHR first-character, integer-array(*)

where: first-character = string-expression

Sample:

```
ALLOCATE INTEGER Onechar(1:CHRY,1:CHRX)
SET CHR 65,Onechar(*)
ALLOCATE INTEGER Several(1:5;1:CHRY,1,:CHRX)
SET CHR 66,Several(*)
```

Description:

This command can be used to redefine the appearance of one or more characters. The computer display must support redefinition of alpha characters or an error will be returned.

This statement is not supported by HTBasic.

See Also:

CHRX, CHRY, SYMBOL

SET DISPLAY MASK

Specifies which planes can be seen on the alpha display.

Syntax: SET DISPLAY MASK numeric-expression

Sample:

```
SET DISPLAY MASK Visible
SET DISPLAY MASK IVAL("1010",2)
```

Description:

The numeric expression value specifies which display bit planes are displayed. This statement does not affect monochrome displays. This statement is equivalent to CONTROL CRT,20;m.

This statement is not supported by HTBasic. Use MERGE ALPHA or SEPARATE ALPHA.

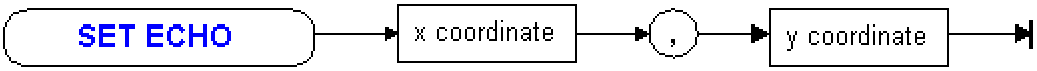
See Also:

ALPHA HEIGHT, ALPHA PEN, CLEAR SCREEN, MERGE ALPHA, SEPARATE ALPHA, SET ALPHA MASK

SET ECHO

Sets the echo location on the PLOTTER IS device.

Syntax: SET ECHO x-coordinate,y-coordinate



Sample:
```
SET ECHO Xx,Yy
SET ECHO 120,240
```

View Sample: **SET ECHO.BAS** **(also found in examples directory)**

Description:

The **SET ECHO** statement specifies a location for the PLOTTER IS echo indicator. If the PLOTTER IS device is a display, the echo is a cross-hair. If the PLOTTER IS device is a plotter, the echo is the pen or device pointer.

The cross-hair is displayed at the specified location if it is within the device limits. If the specified location is outside the device limits the cross-hair is not displayed. Thus, to turn off the cross-hair, specify a position off screen.

The plotter pen is moved (with the pen up) to the specified location if it is within the clip limits. If the specified location is outside the clip limits the pen is moved to and then along the clip limit.

The location returned by the READ LOCATOR statement can be used with the **SET ECHO** statement to cause the echo to track the GRAPHICS INPUT IS location.

Use SET LOCATOR to specify a new GRAPHICS INPUT IS location.

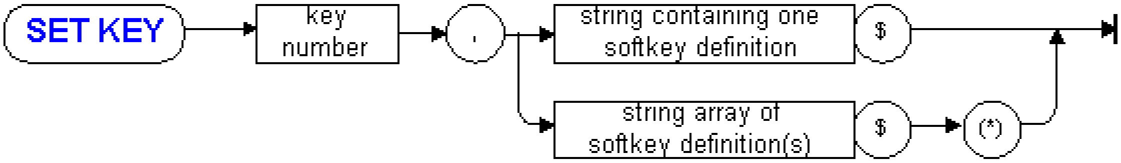
See Also:

DIGITIZE, GRAPHICS INPUT IS, PLOTTER IS, READ LOCATOR, SET LOCATOR, TRACK, WHERE

SET KEY

Defines one or more softkey macros.

Syntax: SET KEY key-number, {string-expression | string-array$(*)}


SET KEY
key number
,
string containing one softkey definition
$
string array of softkey definition(s)
$
(*)

Sample: `SET KEY 2,Keytwo$`
`SET KEY First_key,Several_keys$(*)`

View Sample: **SET KEY.BAS** **(also found in examples directory)**

Description:

Softkey macros may be defined with the **SET KEY** statement. The key-number is a numeric expression which is rounded to an integer and should be in the range zero through twenty-three. If a string expression is specified, then only one key is defined. If a string array is specified, then successive keys, starting with the key-number specified, are defined from the elements of the string array.

Once defined, the key definition is displayed in the softkey menu. Pressing the softkey (when no ON KEY is defined for that key) will type the characters specified in the definition, just as if they had been typed on the keyboard. The definition can include function keys, such as CLEAR SCR.

If the definition begins with a CLR LN key (CHR$(255) & "#"), only the characters after the CLR LN will be displayed. If the definition begins with a CONTINUE key, the two characters (CHR$(255) & "C") will be replaced with the string "CONTINUE". If the definition begins with a RUN key, the two characters (CHR$(255) & "R") will be replaced with the string "RUN".

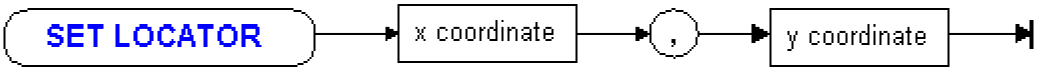
See Also:

EDIT KEY, LIST KEY, LOAD KEY, READ KEY, SCRATCH KEY, STORE KEY

SET LOCATOR

Sets a new graphic locator position on the GRAPHICS INPUT IS device.

Syntax: SET LOCATOR x-position,y-position



Sample: `SET LOCATOR 20,30`

`SET LOCATOR Xx,Yy`

View Sample: **SET LOCATOR.BAS** **(also found in examples directory)**

Description:

The **SET LOCATOR** statement specifies the current location for the GRAPHICS INPUT IS device. Subsequent movement of the GRAPHICS INPUT IS device will be relative to the new location specified. **SET LOCATOR** only works with graphic input devices that use relative locators (i.e. mouse, arrow-keys) and not with those that use absolute locators (i.e. tablets).

Use SET ECHO to specify a new PLOTTER IS echo location.

See Also:

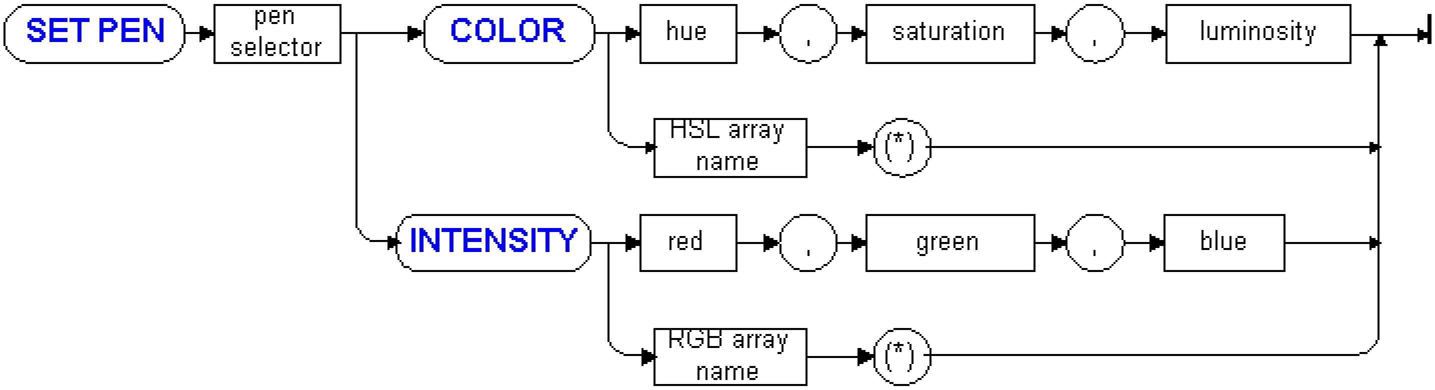
DIGITIZE, GRAPHICS INPUT IS, READ LOCATOR, SET ECHO, TRACK, WHERE

SET PEN

Defines part or all of the color map.

Syntax: SET PEN pen-number COLOR { h, s, l | numeric-array(*) }
SET PEN pen-number INTENSITY { r, g, b | numeric-array(*) }

where: h,s,l, r,g,b = numeric-expressions in the range zero to one.



Sample:
```
SET PEN Num COLOR H,S,L
SET PEN Crayons COLOR Hslarray(*)
SET PEN Name INTENSITY Red,Green,Blue
SET PEN 1 INTENSITY 3/15,5/15,9/15
```

View Sample: **SET PEN.BAS** **(also found in examples directory)**

Description:

The **SET PEN** statement changes the color map values used for each available pen number. A color may be specified in either RGB or HSL color space (see COLOR for an explanation about RGB and HSL color spaces). Redefine multiple pens using the array specifier or redefine individual pens by specifying one HSL or RGB color value. In either case, the pen-number specifies the first entry in the color map to be defined. The pen-number is a numeric expression which is rounded to an integer and should be in the range 0 to n-1, where n is the number of colors.

The closest possible color will be used if the computer display cannot display the color you select. When drawing an area in a certain color, it may be possible to produce the color more accurately by specifying **SET PEN** followed by AREA PEN, rather than specifying AREA COLOR or AREA INTENSITY.

Any pixels already drawn with the specified pen are changed to the new. All **SET PEN** statements take effect immediately upon execution. The effects of all **SET PEN** statements last until the next **SET PEN** statement of the same type, or until GINIT or QUIT. In cases where dithering is used, changing the color map changes the colors available to the dithering process.

Array

If an array is used to set more than one pen, the array must be 2-dimensional and have 3 columns. The number of rows determines the number of pens set. For example,

```
1!RE-SAVE"WINCLR.BAS"
10    PLOTTER IS CRT,"INTERNAL";COLOR MAP
20    DATA .5,.5,.5              ! 8  = dark grey
30    DATA .75,.75,.75           ! 9  = light grey
40    DATA .5, 0, 0              ! 10 = dark red
50    DATA .5,.5, 0              ! 11 = dark yellow
60    DATA 0,.5, 0               ! 12 = dark green
70    DATA 0,.5,.5               ! 13 = dark cyan
80    DATA 0, 0,.5               ! 14 = dark blue
90    DATA .5, 0,.5              ! 15 = dark magenta
180   !
190   DIM Palette(8:15,1:3)
```

```
200   READ Palette(*)
210   SET PEN 8 INTENSITY Palette(*)
220   END
```

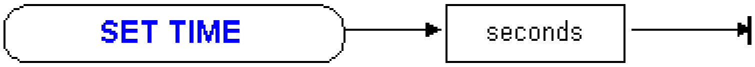
See Also:

AREA PEN, COLOR, PEN, PLOTTER IS

SET TIME

Sets the time of day clock.

Syntax: SET TIME seconds



Sample:

```
SET TIME 43200
SET TIME Hrs*3600+Min*60
```

View Sample: **SET TIME.BAS** **(also found in examples directory)**

Description:

This command sets the time, but not the date. The seconds value is a numeric expression which specifies the number of seconds past midnight. If it includes a fraction, the fraction is rounded to match the clock hardware of the system you are using. The time can be specified to the nearest hundredth of a second, although the PC clock is only accurate to 1/18th of a second.

NT Usage Notes

To set the time, you must have the "Change the system time" user right or belong to a group that has this right. Usually the Administrators and Power Users groups have this right.

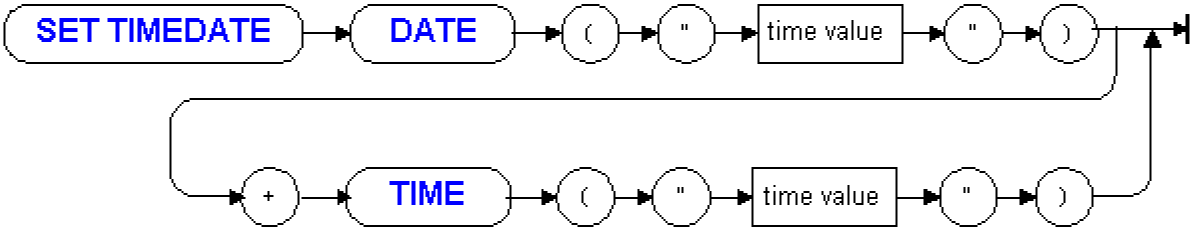
See Also:

DATE, DATE$, TIME, TIME$, SET TIMEDATE, TIMEDATE

SET TIMEDATE

Sets the date and time of the computer's clock.

Syntax: SET TIMEDATE time-value



Sample:

```
SET TIMEDATE TIMEDATE+3600
SET TIMEDATE DATE("6 Nov 1959")
SET TIMEDATE DATE("17 Sep 1987")+TIME("10:00:00")
```

View Sample: **SET TIMEDATE.BAS** **(also found in examples directory)**

Description:

The time-value is a numeric expression and represents a time and date. Use the DATE and TIME functions to convert a time expressed in the familiar formats to the time-value required by this command. If the DATE function is used and the TIME function is not, the time is set to midnight of that date. The date must be within the legal range supported by your operating system. The time may include a fraction, in which case it is rounded to match the clock hardware of the system you are using. The time can be specified to the nearest hundredth of a second, although the PC clock is only accurate to 1/18th of a second. The legal range of dates is 1 Jan 1980 to 31 Dec 2099.

NT Usage Notes

To set the time, you must have the "Change the system time" user right or belong to a group that has this right. Usually the Administrators and Power Users groups have this right.

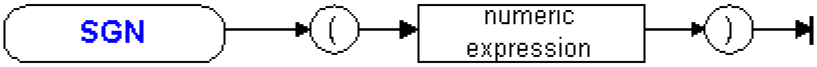
See Also:

DATE, DATE$, TIME, TIME$, SET TIME, TIMEDATE

SGN

Returns the arithmetic sign of an expression.

Syntax: SGN (numeric-expression)



Sample: `Xsgn=SGN(X)`

```
Discriminate=SGN(B*B-4*A*C)
```

View Sample: **SGN.BAS** **(also found in examples directory)**

Description:

SGN returns a value of 1 if the numeric expression is positive, a value of -1 if it is negative and 0 if it is zero.

See Also:

ABS, FRACT, INT, MAXREAL, MINREAL, MOD, MODULO

SHIFT

Shifts a 16 bit binary value.

Syntax: SHIFT(numeric-expression, distance)

where: distance = numeric-expression rounded to an integer.

Sample:

```
Check=SHIFT(Word1,Place)
K=SHIFT(100,-6)
```

View Sample: SHIFT.BAS **(also found in examples directory)**

Description:

The numeric expression is rounded to an integer. The resulting integer, in binary form, is shifted the specified distance. The distance must be in the range ±15. If the distance is positive, bits are moved to the right. Any bits moved out of the right-most bit (the least significant bit) are discarded and zero bits are shifted into the left-most bit (the most significant bit). If the distance is negative, bits are moved to the left. Any bits moved out of the left-most bit are discarded and zero bits are shifted into the right-most bit.

For **SHIFT**(100,5) the number 100 is treated as a binary number and is shifted right five bits as follows:

100	= 0000000001100100
SHIFT(100,5)	= 0000000000000011

The result is returned as the decimal integer, 3.

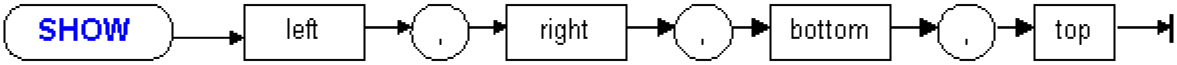
See Also:

BINAND, BINCMP, BINEOR, BINEQV, BINIMP, BINIOR, BIT, ROTATE

SHOW

Defines the graphics unit-of-measure isotropically.

Syntax: SHOW left,right,bottom,top



Sample: `SHOW -10,20,0,75`

```
SHOW Left,Right,Bottom,Top
```

Description:

SHOW, like WINDOW, specifies the values to be displayed within the VIEWPORT or the hard-clip boundaries. They can be any units of measure you wish to work with (inches, miles, years, etc.).

The **SHOW** and WINDOW statements differ in how they map data onto the viewport. **SHOW** uses isotropic units (the X and Y units are of equal length); whereas WINDOW may use non-isotropic units (the X and Y units are of different lengths).

A **SHOW** image can be "mirrored" about the X or Y axes by reversing the order of the limits for each dimension by specifying the high value before the low value.

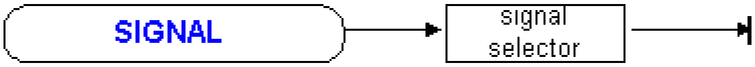
See Also:

CLIP, VIEWPORT, WINDOW

SIGNAL

Initiates a software interrupt.

Syntax: SIGNAL signal-number



Sample: `SIGNAL Post`

`SIGNAL 15`

Description:

The signal number may be a numeric expression which is rounded to an integer and should be in the range of zero through fifteen. If an ON SIGNAL statement has defined a branch for this signal number and the priority allows, the branch is executed.

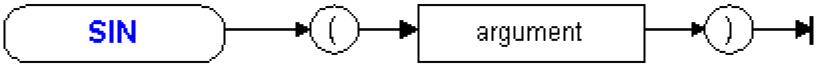
See Also:

DISABLE, ENABLE, OFF SIGNAL, ON SIGNAL

SIN

Returns the sine of the argument.

Syntax: SIN(argument)



Sample: `A=SIN(B)`
`Sine=SIN(Angle)`

View Sample: **SIN.BAS (also found in examples directory)**

Description:

The range of the sine function is -1 to 1 inclusive. The numeric expression is treated as an angle in the current trigonometric mode: RADians or DEGrees. The default trigonometric mode is radians.

COMPLEX Arguments

SIN accepts either a COMPLEX or REAL argument and returns a value of the same type. For COMPLEX arguments the angle must be specified in radians, regardless of the current trigonometric mode. The real and imaginary parts of **SIN**(Z) are calculated (using real arithmetic) as:

REAL(SIN(Z)) = SIN(REAL(Z))*COSH(IMAG(Z))
IMAG(SIN(Z)) = COS(REAL(Z))*SINH(IMAG(Z))

Notice that intermediate values generated during the calculation of the function can cause over or underflow errors for very large or small values of Z.

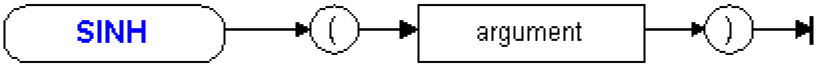
See Also:

ACS, ASN, ATN, COS, TAN, ASNH, ACSH, ATNH, COSH, SINH, TANH, DEG, PI, RAD

SINH

Returns the hyperbolic sine of an expression.

Syntax: SINH(argument)



Sample: `I=SINH(Z)`

`Hsine=SINH(Angle)`

View Sample: **SINH.BAS** **(also found in examples directory)**

Description:

SINH accepts either a COMPLEX or REAL argument and returns a value of the same type. The argument must be specified in radians, regardless of the current trigonometric mode. The real and imaginary parts of **SINH**(Z) are calculated (using real arithmetic) as:

REAL(SINH(Z)) = SINH(REAL(Z))*COS(IMAG(Z))
IMAG(SINH(Z)) = COSH(REAL(Z))*SIN(IMAG(Z))

Notice that intermediate values generated during the calculation of the function can cause over or underflow errors for very large or small values of Z.

See Also:

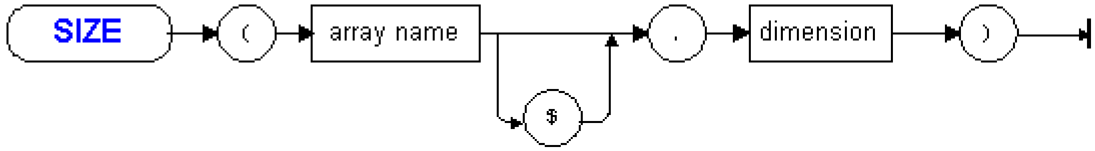
ACSH, ASNH, ATNH, COSH, TANH

SIZE

Returns the number of elements of an array dimension.

Syntax: SIZE(array-name[$],dimension)

where: dimension = integer between 1 and 6, ≤ RANK of array



Sample: `SIZE(A$,X)`

```
Total=SIZE(S$,1)
Upper=BASE(Z,2)+SIZE(Z,2)-1
```

View Sample: **SIZE.BAS** **(also found in examples directory)**

Description:

The **SIZE** is the difference between the upper and lower bounds plus one. The dimension argument may be a numeric expression which is rounded to an integer and should be in the range of one through six. If the array does not have as many dimensions as the dimension you specify, an error will be generated.

See Also:

BASE, DIM, MAXLEN, RANK

SOUND

Produces tones on the computer speaker.

Syntax: SOUND numeric-array(*)
SOUND voice-number, frequency, volume, duration

where: voice-number, frequency, volume, duration = numeric-expressions

Sample:

```
SOUND Voice,Freq,Vol,Dur
SOUND 2,440,10,0.70
SOUND Maryhadalittle(*)
```

Description:

This statement is not supported by HTBasic. Use BEEP

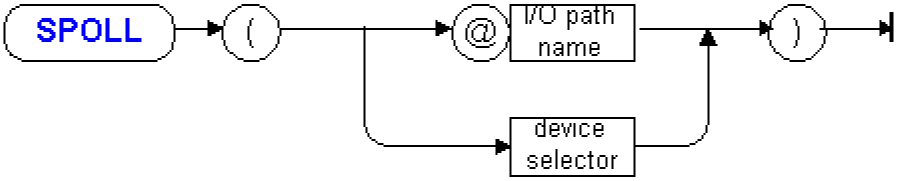
See Also:

BEEP

SPOLL

Performs a serial poll of a IEEE-488 device.

Syntax: SPOLL({@io-path | device-selector})



Sample: `Stat=SPOLL(712)`
`SPOLL(@Dev)`

Description:

The **SPOLL** function returns the integer serial poll response of the specified IEEE-488 device. The computer must be the active controller and a primary device address must be specified. One secondary address may be specified.

The IEEE-488 bus action is: ATN, UNL, MLA, TAD, SPE not-ATN, Read data byte, ATN, SPD, UNT.

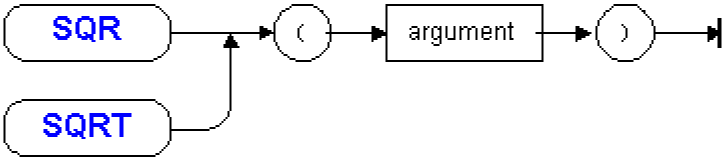
See Also:

ABORT , CLEAR, LOCAL, PASS CONTROL, PPOLL, REMOTE, REQUEST, SEND, TRIGGER

SQRT

Returns the square root of an expression.

Syntax: SQRT(numeric-expression)
SQR(numeric-expression)



Sample:
```
Root=SQRT(10*X)
PRINT "Square Root of";Y;"=";SQR(Y)
```

View Sample: **SQRT.BAS** **(also found in examples directory)**

Description:

The square root function may be entered as either **SQRT** or **SQR**.

COMPLEX Arguments

SQRT accepts either a COMPLEX or REAL argument and returns a value of the same type. **SQRT**(Z) returns the principal value, defined (in real arithmetic) as:

REAL(SQRT(Z)) = SQRT((SQRT(REAL(Z)^2+IMAG(Z)^2)+REAL(Z))/2)

IMAG(SQRT(Z)) = SGN(Y)*
SQRT((SQRT(REAL(Z)^2+IMAG(Z)^2)-REAL(Z))/2)

which returns a real part = 0. The domain of **SQRT** includes all points in the complex plane. However, intermediate values generated during the calculation of the function can also cause over or underflow errors for very large or small values of Z.

See Also:

EXP, LOG, LGT

STATIC

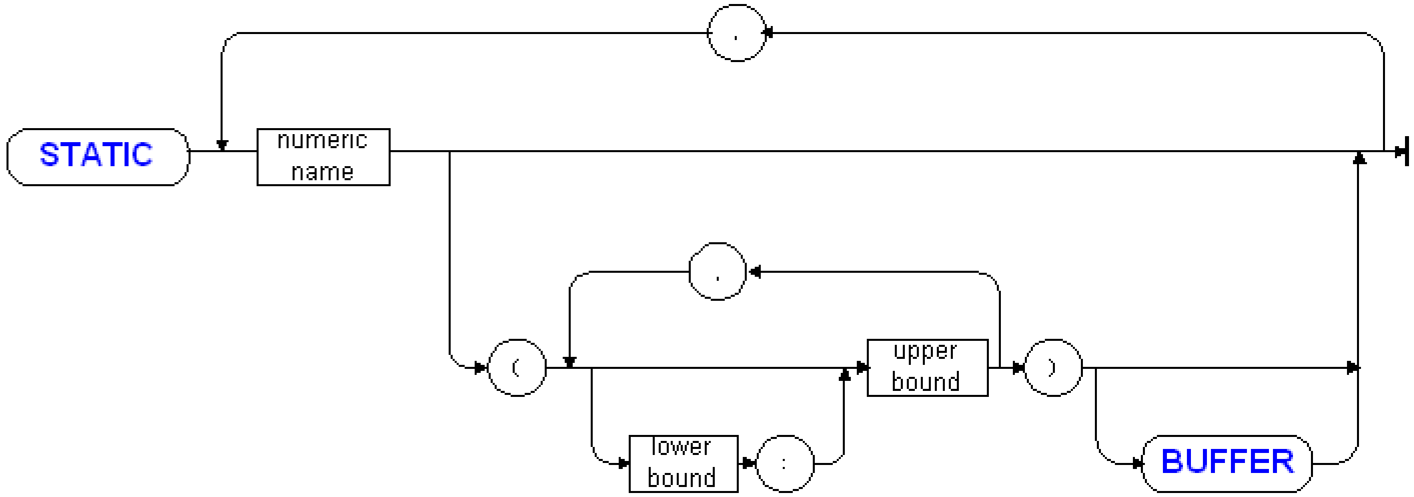
Reserves storage for STATIC variables and arrays.

Syntax:

STATIC *item* [,*item*...]

Where:

item = numeric-name[(*bounds*)] [=initial numeric value] [BUFFER] |
string-name$[(*bounds*)]’[’*length*’]’ [=initial string value] [BUFFER]
bounds = [lower-bound:]upper-bound [,*bounds*...]



Sample:

```
STATIC INTEGER Myint
STATIC REAL Myreal=5.7
STATIC INTEGER Intarray(100)
STATIC LONG Longarray(100)=10
STATIC A1$[100] BUFFER
STATIC A2$[100]=”This is a test”
STATIC A3$(100)[100]
STATIC A4$(100)[100]=”This is a test” BUFFER
STATIC INTEGER J1,J2=1,J3(100,200,400,500,600),LONG
  J4(100)=0 BUFFER
```

Description:

STATIC is a data condition. STATIC variables are persistent during a single run of an HTBasic program. Typically, STATIC variables will only be used in SUB programs and/or FN functions because the MAIN context is usually called only once.

STATIC variables can effectively take the place of COM variables as they are presently used in many cases. If access to a COM variable is required in multiple SUBs and/or Functions (DEF FN) and/or the Main context, then a STATIC variable is not appropriate. The scope of a STATIC variable is limited to the context in which it is declared. In other words, a STATIC variable declared in a SUB program cannot be accessed anywhere other than within that particular SUB program, however, it will retain the variable value between calls to that subroutine.

Up to 6 bounds may be specified, the initial values are optional. Specifying an initial value for an array initializes each individual element in all dimensions of the array to the initial value specified.

See Also:

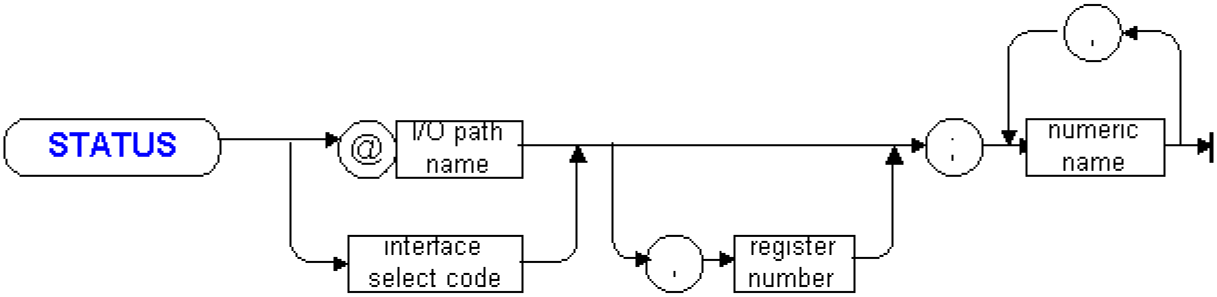
COMPLEX, INTEGER, I/O PATH, LONG, REAL, STRING

STATUS

Returns control information from an interface or I/O path.

Syntax: STATUS source [,register] ;variable [,variable...]
STATUS(source, register)
STATUS @widgethandle;RETURN(return attribute list)

where: source = @io-path | interface-select-code
register = numeric-expression rounded to an integer
variable = numeric-name [(*)]



Sample:

```
STATUS CRT;Col,Row
STATUS @Io,1;Type
IF STATUS(CRT,6) THEN ALPHA OFF
PRINT "Baud rate is ";STATUS(9,3)
STATUS 1801,19;Gains(*)
STATUS @Field1;RETURN (“VALUE”:Limit$)
STATUS @Input;RETURN (“VALUE”:Setpoint)
STATUS @Meter1;RETURN (“LOW LIMIT”:Low_lim,
“HIGH LIMIT”:High_lim)
```

View Sample: STATUS.BAS (also found in examples directory)

Description:

The I/O path or interface register contents are copied into the numeric variables, starting at the specified register number and continuing until the variable list is exhausted. The default register number is zero.

The range of legal registers and the meaning of values read from them differ for each interface. The *User's Guide* describes the CONTROL and **STATUS** registers for each interface and for I/O paths. Typically, registers return integer values and if you specify real values, they are rounded to integers. However, some drivers return real values or even arrays, so the documentation should be consulted.

The function form of **STATUS** complements the **STATUS** statement. It allows immediate access to a single register without need for a temporary variable or separate **STATUS** statement. However, the **STATUS** function can only return one value at a time, while the **STATUS** statement can return multiple registers in a single statement.

Each widget has a variety of attributes that control its appearance and behavior. The **STATUS** statement is used to query the value of a widget attribute. The widget must have been created previously using an ASSIGN statement. Attributes are either scalar (may contain a single value) or vector (may be assigned an array of values) and have values of either numeric or string type.

A shorthand method is available that permits you to query values of several attributes without naming them individually on the **STATUS** statement. (Only scalar attributes may be queried with this shorthand method.)

- You store all the attributes in a string array and supply an array to receive attribute values.

- Then, when you supply the array names to the **STATUS** statement, the value of each attribute named in each element of the attribute array will be returned in the corresponding element of the value array.

- Elements of the attribute array that contain nothing, or nothing but blanks, will be ignored and the corresponding element of the value array will remain unchanged.

Since widget handles are equivalent to I/O path names, you may use the **STATUS** statement to query the value of registers, which provide information about the widget. For widgets, Status Register 0 and Status Register 1 are defined.

Status Register 0 is defined for all I/O paths. For example:

```
STATUS @Io_path,0;Numeric_var
```

For widgets, this returns a 5 to numeric_var (5 means @Io_path is a widget). Status Register 1 is defined for all I/O paths assigned to a device. For example:

```
STATUS @Pb_12,1;Numeric_var
```

For widgets, this will return a 6 to numeric_var (6 means @Pb_12 is a device associated with the internal graphics CRT).

Any status register greater than 1 will cause *Error 155 - Bad interface register number.* Using ENTER, OUTPUT, TRANSFER, etc., (all other commands associated with I/O paths assigned to devices) generates *Error 170 - I/O operation not allowed.*

Porting to HP BASIC:

STATUS @Iopath,2 always returns a 4. **STATUS** @File,3 returns the current length, not the CREATE length. This is because files are extendible under DOS and Windows.

The **STATUS**() function is an addition to HTBasic. Any **STATUS** or CONTROL registers greater than 99 are also additions. As in HP BASIC, **STATUS** register 0 of interface cards contains the card ID. Interface cards that are available on a PC, but not on an HP BASIC Workstation are identified with ID numbers greater than or equal to 300. These new features should not be used in programs that must be ported back to HP BASIC.

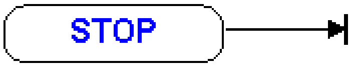
See Also:

CONTROL, READIO, WRITEIO

STOP

Terminates program execution.

Syntax: STOP



Sample:

```
STOP
IF Finish THEN STOP
```

View Sample: **STOP.BAS** **(also found in examples directory)**

Description:

When **STOP** is encountered, the program quits execution, I/O paths not in COM are closed and all variables are discarded. CONT cannot be used after **STOP**. To restart the program you must use the RUN statement. Use PAUSE to temporarily halt program execution and CONT to resume program execution.

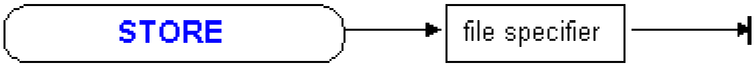
See Also:

CONT, PAUSE, RUN

STORE

Stores the BASIC program in a file.

Syntax: STORE file-specifier



Sample: `STORE Vol$&Name$`
`STORE "Fullprg"`

View Sample: **STORE.BAS** **(also found in examples directory)**

Description:

A new file of type PROG is created and the BASIC program currently in memory is written to the file in binary form. If the file already exists, an error is reported. Use RE-STORE to update an existing file. Use LOAD to re-enter the program into the computer.

Porting to HP BASIC:

HP BASIC PROG files and HTBasic PROG files are not compatible. To move programs between the two environments, use ASCII program files.

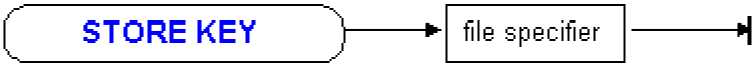
See Also:

GET, LIST, LOAD, RE-SAVE, RE-STORE, SAVE, STORE KEY

STORE KEY

Stores the softkey definitions in a file.

Syntax: STORE KEY file-specifier



Sample: `STORE KEY Path$&"MACROS.HTB"`

`STORE KEY "/usr/htb/keys"`

View Sample: **STORE KEY.BAS** **(also found in examples directory)**

Description:

A new file of type **BDAT** is created with the name specified. If the file already exists, an error is reported. Use RE-STORE KEY to update an existing file.

Using FORMAT OFF, the definition for any defined softkey is written to the file by outputting two items. The first item is an integer, specifying the key number. The second item is a string, giving the key definition. Use LOAD KEY to re-enter the softkey macros into the computer.

FORMAT MSB FIRST is used to write the file. This makes key definitions compatible with HP Workstations and can easily be used with HP BASIC.

See Also:

EDIT KEY, LIST KEY, LOAD KEY, READ KEY, RE-STORE KEY, SCRATCH KEY, SET KEY

STORE SYSTEM

Stores BASIC and loaded BINs into a file.

Syntax: STORE SYSTEM file-specifier

Sample:

```
STORE SYSTEM "Full"
```

Description:

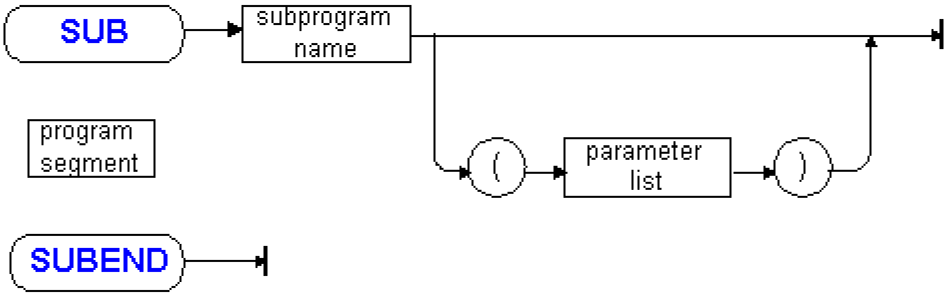
In HP BASIC, this statement stores a copy of the operating system with all loaded BINs already linked in. Under HTBasic, this statement is not supported. Use the HTBasic AUTOST file to load HTBasic device drivers.

SUB

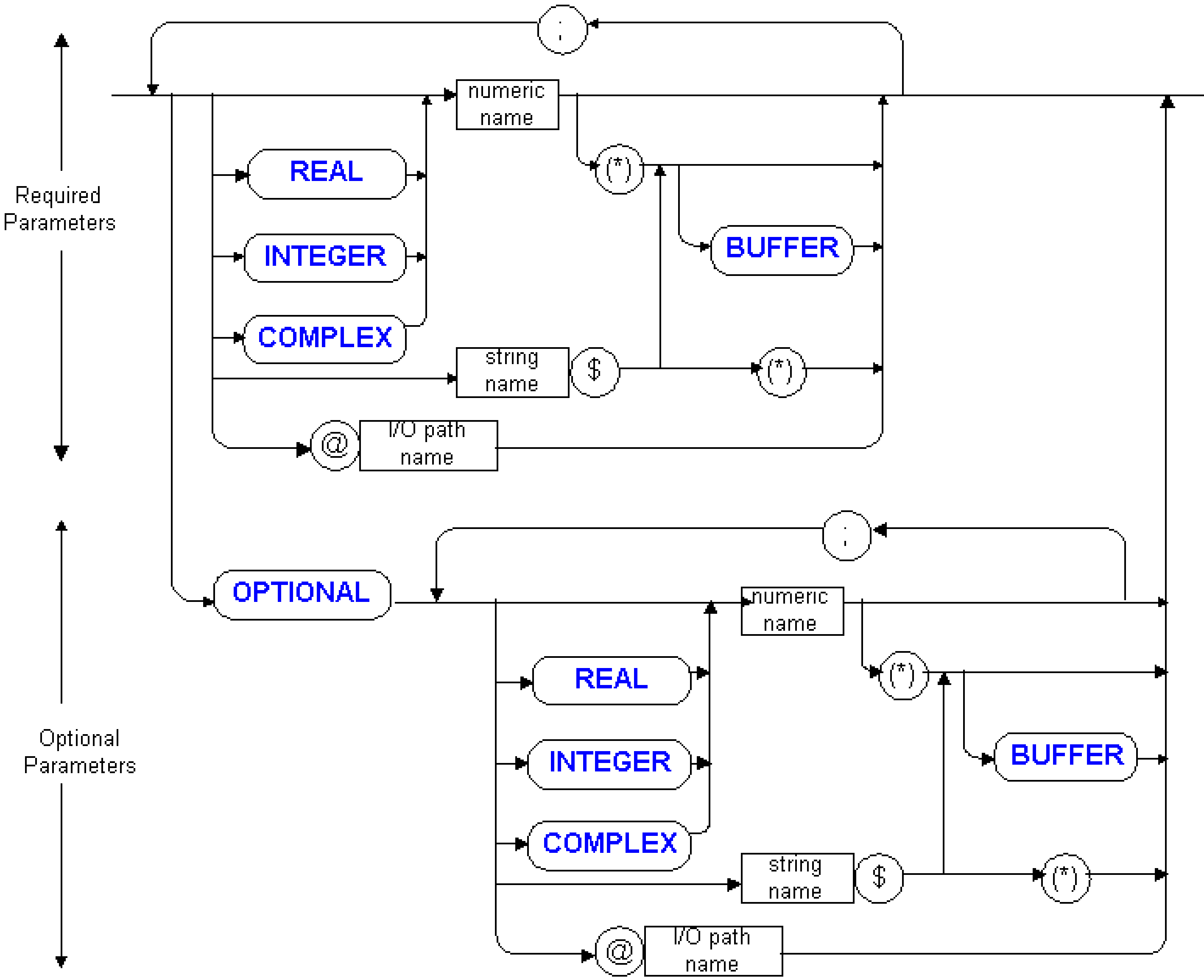
Defines a subprogram and specifies formal parameters.

Syntax: SUB subprogram-name [(parameter-list)]
statements
[SUBEXIT]
statements
SUBEND

where: statements = zero, one or more program statements
including additional SUBEXIT statements
parameter-list = [param [,param...]] [,] [OPTIONAL param [,param...]]
[,] = the optional comma is only needed when items
occur on both sides of it
param = [REAL|INTEGER|COMPLEX] numeric-name [(*)[BUFFER]] |
string-name$ [(*) | BUFFER] | @io-path



parameter list:



Sample: `SUB Unit1`

```
SUB Link(String$)
SUB Procm(INTEGER Array(*),OPTIONAL @Lpr,Name$)
SUB Plot(Buff$ BUFFER,Coor)
```

View Sample: **SUB.BAS** **(also found in examples directory)**

Description:

SUB subprograms must follow the MAIN program's END statement. The first line must be a **SUB** statement and the last line a **SUBEND** statement. The lines between **SUB** and **SUBEND** statements define a subprogram which can be called by other parts of the program with the CALL statement.

Unless the **OPTIONAL** keyword is specified, the number of CALL arguments must match the number of **SUB** parameters; each argument must be of the same type (numeric or string) as the corresponding parameter. Any parameters to the right of the **OPTIONAL** keyword are optional in the CALL statement. NPAR returns the number of arguments in the current CALL statement. All variables defined in a subprogram that are not COM variables are local to the subprogram. Upon each entry to the subprogram they are set to zero.

A CALL to a subprogram, transfers control to the first statement of that subprogram and starts executing from there. Execution proceeds normally until either a **SUBEND** or **SUBEXIT** statement is executed, at which point control returns to the statement after the CALL. The **SUBEXIT** statement allows a return from the subprogram at points other than the **SUBEND**. Multiple **SUBEXIT**s are allowed in a subprogram. **SUBEXIT** may appear in an IF statement, **SUBEND** can not.

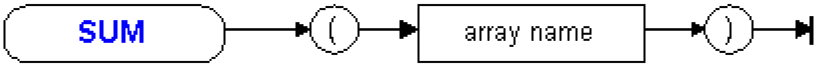
See Also:

CALL, DEF FN, FN

SUM

Returns the sum of all elements in a numeric array.

Syntax: SUM(numeric-array)



Sample: `S1=SUM(A2)`

```
PRINT SUM(Array)
```

View Sample: **SUM.BAS** **(also found in examples directory)**

Description:

If the array has type REAL, then **SUM** returns a REAL value. If the array has type INTEGER, then **SUM** returns an INTEGER value and the possibility of INTEGER overflow exists during the summing of an array.

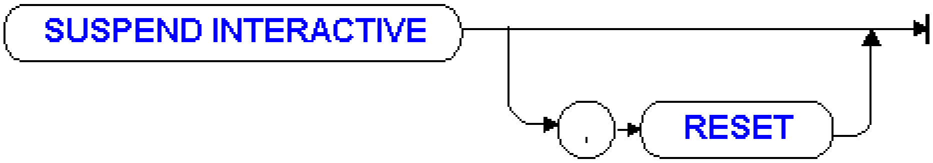
See Also:

CSUM, RSUM

SUSPEND INTERACTIVE

Deactivates program control keys.

Syntax: SUSPEND INTERACTIVE [,RESET]



Sample: `SUSPEND INTERACTIVE,RESET`

View Sample: **SUSPEND INTERACTIVE.BAS** **(also found in examples directory)**

Description:

The normal functions of the program control keys CLR I/O, ENTER, PAUSE, STEP and STOP, are disabled. The RESET key may also be disabled by specifying the optional **RESET** keyword. The keys are only disabled while the program is running.

RESUME INTERACTIVE, END, GET, LOAD, RUN, SCRATCH or STOP will re-enable the program control keys as well as the RESET key.

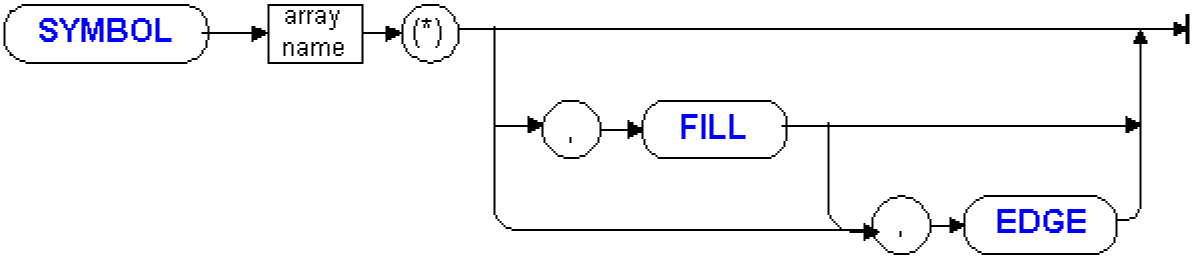
See Also:

RESUME INTERACTIVE

SYMBOL

Allows the user to define label symbols.

Syntax: SYMBOL numeric-array(*) [,FILL] [,EDGE]



Sample: `SYMBOL Code(*)`
`SYMBOL Hieroglyph(*),FILL,EDGE`

View Sample: **SYMBOL.BAS** **(also found in examples directory)**

Description:

SYMBOL uses a two-dimensional two-column or three-column array to plot a User-defined symbol. They are created with moves and draws in the LABEL font coordinate system, an area nine units wide and fifteen units high. Unlike LABEL, **SYMBOL** allows coordinates outside the character cell.

The symbol is drawn using the current pen control and line type and will be clipped at the clip boundary. A move is always done to the first point and the current pen position is left at the last X,Y position specified in the array and is not updated to the next character position. The CSIZE, LDIR and LORG statements affect the **SYMBOL** statement.

See PLOT for an explanation of **FILL**, **EDGE**, and array operations supported by **SYMBOL**. See the *User's Guide* for more information about the **SYMBOL** coordinate system.

Porting to HP BASIC:

LORG 5 moves the symbol origin from (0,0) to (5,8). In HP BASIC it moves the origin to (4.5,7.5).

See Also:

CSIZE, LABEL, LDIR, LORG, PEN, PLOT, SET CHR

SYSBOOT

Reboots the computer.

Syntax: SYSBOOT

Sample:

```
SYSBOOT
```

Description:

HTBasic does not support **SYSBOOT**, which under HP BASIC reboots the computer. Since HTBasic runs as a guest of an operating system, it is considered inappropriate to reboot the computer. Under some operating systems, rebooting the computer inappropriately can cause loss of data. To terminate HTBasic, use the QUIT ALL statement.

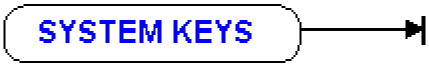
See Also:

EXECUTE, QUIT ALL

SYSTEM KEYS

Displays the System Softkeys Menu.

Syntax: SYSTEM KEYS



Sample:

```
SYSTEM KEYS
IF Menu THEN SYSTEM KEYS
```

View Sample: **SYSTEM KEYS.BAS** **(also found in examples directory)**

Description:

This statement has no effect if KBD CMODE is on. This statement is equivalent to CONTROL KBD,2;0.

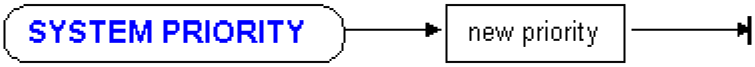
See Also:

KBD CMODE, KEY LABELS, KEY LABELS PEN, USER KEYS

SYSTEM PRIORITY

Sets the system priority to a specified level.

Syntax: SYSTEM PRIORITY priority



Sample:

```
SYSTEM PRIORITY Degree
SYSTEM PRIORITY 2
```

View Sample: **SYSTEM PRIORITY.BAS** **(also found in examples directory)**

Description:

The priority may be a numeric expression and is rounded to an integer in the range of zero (the lowest priority) through fifteen (the highest priority). The default priority is zero. ON END, ON ERROR, and ON TIMEOUT have higher priorities than the highest user **SYSTEM PRIORITY**.

Any events defined at an equal or lower priority will be logged and not executed until the system priority is lowered.

If the system priority is changed within a subprogram, it will be restored when the subprogram ends.

See Also:

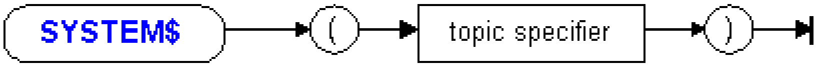
ON, SYSTEM$("SYSTEM PRIORITY")

SYSTEM$

Returns system status and configuration information.

Syntax: SYSTEM$(information)

where: information = a string-expression containing one of the strings from the table below.



Sample:

```
M=VAL(SYSTEM$("AVAILABLE MEMORY"))
PRINT "Version "&SYSTEM$("VERSION:HTBasic")
```

View Sample: **SYSTEM$.BAS** **(also found in examples directory)**

Description:

SYSTEM$** returns system information in a string. The information returned depends on which of the following strings is specified in the **SYSTEM$ command.

Porting to HP BASIC. Minor differences in some **SYSTEM$** responses exist where appropriate to reflect hardware or operating system differences.

AVAILABLE MEMORY

Returns the available memory in bytes. In most cases the FRE function is easier to use. The amount of available memory when HTBasic is started can be specified with a command line switch. See FRE.

CRT ID

Returns a twelve character CRT identification string. A space in a position indicates that capability is not present.

Bytes	Meaning
1	always "6".
2	always ":".
3-5	CRT width, for example " 80".
6	"H" if at least one display enhancement is supported, i.e. inverse, blink, underline. Not all CRTs support all enhancements.
7	"C" if colors are available in at least one screen mode.
8	"G" if graphics are available.
9	"B" if the display is bit-mapped.
10-12	Maximum value for ALPHA PEN.

DISP LINE

The present content of the display line is returned. This allows you to write subroutines that temporarily save off the display line content, DISP something else and then restore the display line.

SYSTEM$("DISP LINE") is a new HTBasic function that is not available in HP BASIC. It should not be used in programs that must be ported back to HP BASIC.

DUMP DEVICE IS

Returns a string specifying the current DUMP DEVICE.

GRAPHICS INPUT IS

Returns a string specifying the current GRAPHICS INPUT IS device.

GFONT IS

Returns a string specifying the current GFONT IS font.

KBD LINE

Returns a string whose content is the same as the current keyboard input line.

KEYBOARD LANGUAGE

Returns a string identifying foreign language keyboards. On some computers, it is not possible for HTBasic to know the keyboard type. On these systems "ASCII" is returned regardless of the actual keyboard.

LEXICAL ORDER IS

Returns the current language set by the LEXICAL ORDER IS command. "ASCII" is the default.

MASS MEMORY

Returns a sixteen character string identifying types and numbers of mass storage devices attached. On some computers, this information is not available to HTBasic. On these systems, "0" is returned for each device type. If the number of devices of any type exceeds nine, "*" is returned in that byte position.

Bytes	**Meaning**
1	number of internal disk drives.
2-4	not assigned.
5	number of initialized EPROM cards (always 0).
6	number of bubble memory cards (always 0).
7-16	not assigned.

MASS STORAGE IS or MSI

Returns the current device and directory. **MSI** is an abbreviation for MASS STORAGE IS and returns the same information.

PLOTTER IS

Returns a string specifying the current PLOTTER IS device.

PRINTALL IS

Returns a string specifying the current PRINTALL IS device.

PRINTER IS

Returns a string specifying the current PRINTER IS device.

PROCESS ID

Under multitasking operating systems such as NT, this call returns the process ID of HTBasic. Under single-tasking operating systems such as DOS, this call always returns "0".

SERIAL NUMBER

Returns a string containing the serial number. The number is unique for that class of hardware. On a PC, the serial number is an eleven character string read from the ID Module connected to the parallel port. If the serial number can not be found, the string "11111111111" is returned.

SYSTEM ID

A string identifying the hardware system is returned. The DOS Version of HTBasic uses the IBM PC System ID byte located at F000:FFFE to determine what seven character string should be returned. The following table gives the responses generated:

ID Byte	**Computer**	**SYSTEM$("SYSTEM ID")**
F8	PS/2 Model 80	"PS/2 80"
F9	PC Convertible	"PC Conv"
FA	PS/2 Model 30	"PS/2 30"
FB	PC/XT	"PC/XT "
FC	PC/AT, PS/2 Models 50/60	"PC/AT "
FD	PC Jr	"PCjr "
FE	PC/XT	"PC/XT "
FF	PC	"PC "
other	Unknown	"PC "

Under Windows and NT, three numbers are returned, separated by commas. The first number is the processor type, the second is the number of processors and the third is the machine OEM ID, if it has one.

SYSTEM PRIORITY

Returns a string containing the current system priority. Use VAL(**SYSTEM$**("SYSTEM PRIORITY")) to retrieve the priority as a numeric value.

TIMEZONE IS

Under operating systems like Windows, which store the local time in the real time clock, this call always returns "0". Under operating systems like UNIX, which store Greenwich Mean Time in the real time clock, this call returns the number of seconds difference between your local time and GMT. Negative values represent timezones west of Greenwich.

TRIG MODE

Returns the current trigonometric mode, "DEG" for degrees and "RAD" for radians.

VERSION:BASIC

Returns a string containing the HP BASIC version number emulated, i.e., "5.1", "6.2", etc.

VERSION:HTB

Returns a string containing HTBasic version information. This is the same information printed on the first line of the CRT when HTBasic starts.

This function can be useful for programs that run on both HP BASIC and HTBasic systems, enabling them to determine which system they are currently running on. The following example sets a variable according to the system running the program:

```
10  SUB Which_system
20    COM /Which_system/Htbasic,Hpbasic
30    IF SYSTEM$("VERSION:HTB")="0" THEN
40      Hpbasic=1
50    ELSE
60      Htbasic=1
70    END IF
80  SUBEND
```

VERSION:OS

Returns a string containing operating system revision information. The string is of the form "x.yy Windows" and under NT it is "x.yy Windows NT". X is the major revision and yy is the minor revision.

VERSION:*bin-name*

Returns a string containing the version number of the binary named. Replace *bin-name* with the name of the binary of interest. LIST BIN can be used to see the version numbers for all loaded binaries.

WILDCARDS

Returns "OFF:" if wildcarding is turned off. This function always returns "ON:". See WILDCARDS.

WINDOW SYSTEM

Returns "Console" under most versions of HTBasic. Under some versions it returns the name of the current screen driver. See PLOTTER IS for an explanation of screen drivers.

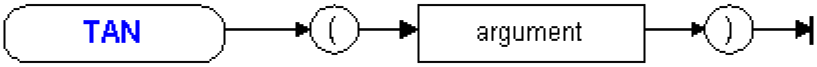
See Also:

DEG, DUMP DEVICE IS, GRAPHICS INPUT IS, PLOTTER IS, LEXICAL ORDER IS, MSI, PRINTALL IS, PRINTER IS, RAD, SYSTEM PRIORITY

TAN

Returns the tangent of an expression.

Syntax: TAN(argument)



Sample:

```
A=TAN(B)
Tangent=TAN(X)
PRINT "Tangent of";Angle;"=";TAN(Angle)
```

View Sample: **TAN.BAS (also found in examples directory)**

Description:

The tangent of an angle is the sine of the angle divided by the cosine of the angle. The numeric expression is treated as an angle in the current trigonometric mode: RADians or DEGrees. The default units are radians. **TAN** is defined for all real numbers except ±PI/2 (±90 degrees) and other odd multiples of PI/2 (90 degrees).

COMPLEX Arguments

TAN accepts either a COMPLEX or REAL argument and returns a value of the same type. For COMPLEX arguments the angle must be specified in radians, regardless of the current trigonometric mode. The real and imaginary parts of **TAN**(Z) are calculated (using real arithmetic) as:

REAL(TAN(Z)) = SIN(2*REAL(Z))/D
IMAG(TAN(Z)) = SINH(2*IMAG(Z))/D

where:

D = COS(2*REAL(Z))+COSH(2*IMAG(Z))

The domain of TANH includes all points in the complex plane except CMPLX(PI/2,0) and other odd multiples of PI/2. Also, intermediate values generated during the calculation of the function can also cause over or underflow errors for very large or small values of Z.

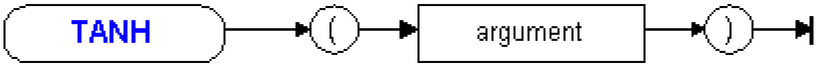
See Also:

ACS, ASN, ATN, COS, SIN, ASNH, ACSH, ATNH, COSH, SINH, TANH, DEG, PI, RAD

TANH

Returns the hyperbolic tangent of an expression.

Syntax: TANH(numeric-expression)



Sample:

```
A=TANH(B)
Htangent=TANH(X)
PRINT "Hyperbolic Tangent of";Angle;"=";TANH(Angle)
```

View Sample: **TANH.BAS** **(also found in examples directory)**

Description:

TANH accepts either a COMPLEX or REAL argument and returns a value of the same type. The argument must be specified in radians, regardless of the current trigonometric mode. The real and imaginary parts of **TANH**(Z) are calculated (using real arithmetic) as

REAL(TANH(Z)) = SINH(2*REAL(Z))/D
IMAG(TANH(Z)) = SIN(2*IMAG(Z))/D

where:

D = COSH(2*REAL(Z))+COS(2*IMAG(Z))

The domain of **TANH** includes all points except CMPLX(0,PI/2+PI*K), where K can be any integer. However, intermediate values generated during the calculation of the function can cause over or underflow errors for very large or small values of Z.

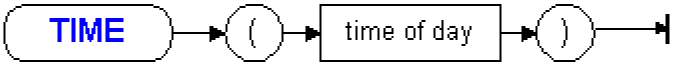
See Also:

ACSH, ASNH, ATNH, COSH, SINH

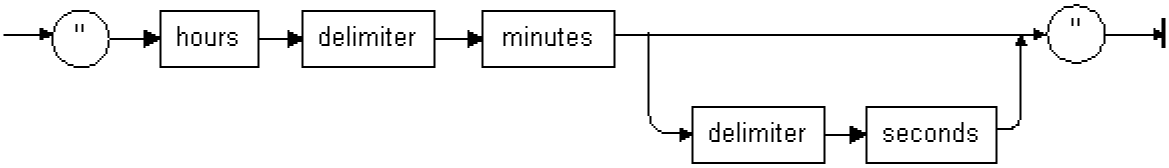
TIME

Converts a time-of-day string to seconds after midnight.

Syntax: TIME(string-expression)



literal form of time of day:



Sample:

```
Seconds=TIME(Clock$)
SET TIME TIME("3:56:30")
ON TIME TIME("17:00") RECOVER Athome
```

View Sample: **TIME.BAS** **(also found in examples directory)**

Description:

A string expression in the form HH:MM[:SS] is converted into an equivalent number of seconds past midnight in the range 0 through 86,399. Leading blanks and non-numeric characters are ignored.

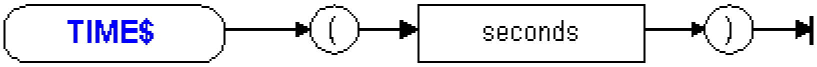
See Also:

DATE, DATE$, TIME$, SET TIME, SET TIMEDATE, TIMEDATE

TIME$

Returns a formatted time of day string.

Syntax: TIME$(numeric-expression)



Sample:

```
PRINT TIME$(TIMEDATE)
Later$=TIME$(Sec+3600)
```

View Sample: **TIME$.BAS** **(also found in examples directory)**

Description:

TIME$** takes a numeric-expression representing seconds past midnight and forms a time of day string with the format HH:MM:SS. If TIMEDATE is used as the argument, then **TIME$ returns the current time of day.

See Also:

DATE, DATE$, TIME, SET TIME, SET TIMEDATE, TIMEDATE

TIMEDATE

Returns the current time and date from the clock.

Syntax: TIMEDATE

Sample:

```
PRINT "The operation took ";TIMEDATE-Start;" seconds"
DISP TIME$(TIMEDATE),DATE$(TIMEDATE)
DISP "Seconds since midnight = ";TIMEDATE MOD 86400
```

View Sample: **TIMEDATE.BAS** **(also found in examples directory)**

Description:

A real number, representing the present time and date is returned. To convert the number to the familiar date and time formats, use TIME$ and DATE$. The value returned is loosely based on the Julian Period, which began in 4713 B.C. To return the current Julian Day, use the following function. Remember that the Julian Day changes at noon.

```
10  DEF FNJd_now
20    RETURN ((TIMEDATE-4300) DIV 86400)-1
30  FNEND
```

See Also:

DATE, DATE$, TIME, TIME$, SET TIME, SET TIMEDATE

TIMEZONE IS

Corrects between GMT and local time for HP BASIC/WS.

Syntax: TIMEZONE IS seconds

where: seconds = numeric-expression

Sample:

```
IF Utah THEN TIMEZONE IS -7*3600
```

Description:

HTBasic does not require this statement and will return an error if an attempt is made to execute it. The editor will allow it to be entered and the syntax checker will check it for correctness to allow you to develop programs and run them under HP BASIC. HP BASIC requires this statement for two reasons: 1) HP BASIC/UX keeps a time clock independent of the UNIX time and 2) it is possible to boot HP BASIC/WS on a computer whose real-time clock is set to Greenwich Mean Time (GMT).

SYSTEM$("TIMEZONE IS") returns the value currently in effect. The offset specifies the difference in seconds between GMT and local time. Negative values specify timezones west of GMT, positive values specify timezones east of GMT. The following table gives offsets in hours for standard time. Multiply the hours given by 3600 before comparing to values returned by SYSTEM$("TIMEZONE IS").

Timezone	Hours
Eastern European	+2
Middle European	+1
Western European	0
Atlantic	-4
Eastern	-5
Central	-6
Mountain	-7
Pacific	-8
Japan	+9
Singapore	+8

See Also:

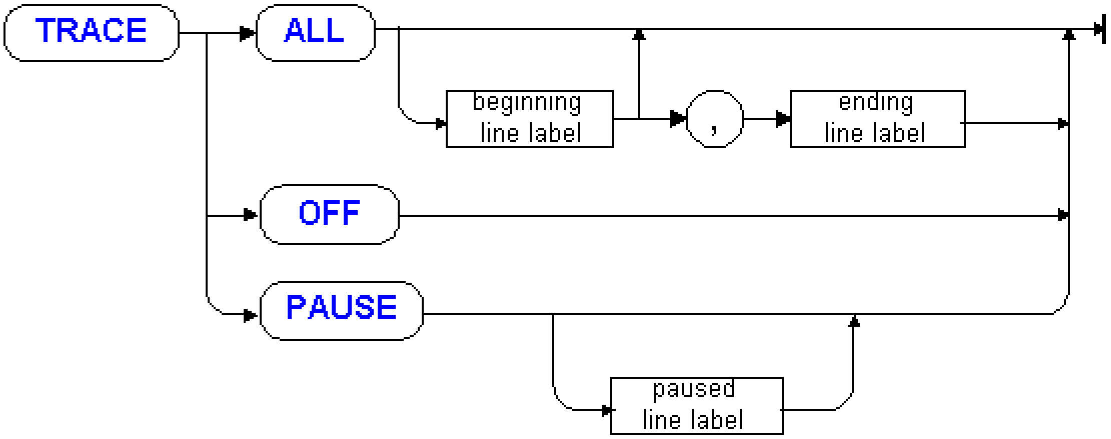
DATE, DATE$, TIME, TIME$, SET TIME, SET TIMEDATE, TIMEDATE

TRACE

Controls the display of information about a running program.

Syntax: TRACE ALL [start-line [,end-line]]
TRACE OFF
TRACE PAUSE [line]

where: line = line-number | line-label



Sample:
```
TRACE ALL 1000,1200
TRACE OFF
TRACE PAUSE 250
```

View Sample: **TRACE.BAS** **(also found in examples directory)**

Description:

TRACE ALL traces program flow and variable assignments. Either the entire program or just a range of program lines may be traced. The trace output is sent to the message line and displays the program line numbers and any modified simple numeric or string variable and its new value. If a full array is modified the entire array is not displayed. If print-all mode is on, then the trace output is also sent to the PRINTALL IS device.

TRACE OFF turns off all tracing functions.

TRACE PAUSE will PAUSE program execution before the specified line and will display the next program line to be executed. If no line is specified, the program pauses before the next line is executed and the current **TRACE PAUSE** line is deactivated. Tracing slows program execution.

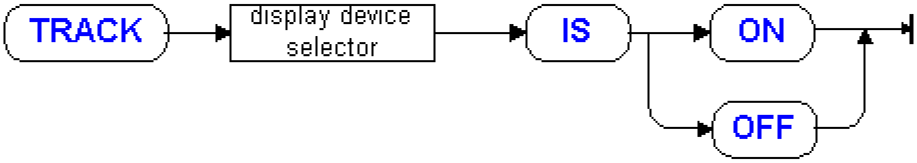
See Also:

CAUSE ERROR, CLEAR ERROR, PRINTALL IS, XREF

TRACK

Enables or disables tracking of the locator position on the display device.

Syntax: TRACK device-selector IS {ON | OFF}



Sample: `TRACK Plot IS ON`

```
TRACK 702 IS OFF
```

View Sample: **TRACK.BAS** **(also found in examples directory)**

Description:

ON enables tracking of the current locator on the PLOTTER IS device during DIGITIZE statements. Tracking stops when a point is digitized and the echo is left at the location of the digitized point. When the display device is a plotter, the pen position tracks the locator. When the CRT is the display device, a crosshair tracks the locator. **OFF** disables tracking of the current locator. To turn off the crosshair, use SET ECHO with coordinates that are off screen

The current locator is defined by a GRAPHICS INPUT IS statement and the current display device is defined by a PLOTTER IS statement. If the device-specifier is not the same as the current PLOTTER IS device, an error is generated.

See Also:

DIGITIZE, GRAPHICS INPUT IS, PLOTTER IS, READ LOCATOR, SET ECHO, SET LOCATOR, WHERE

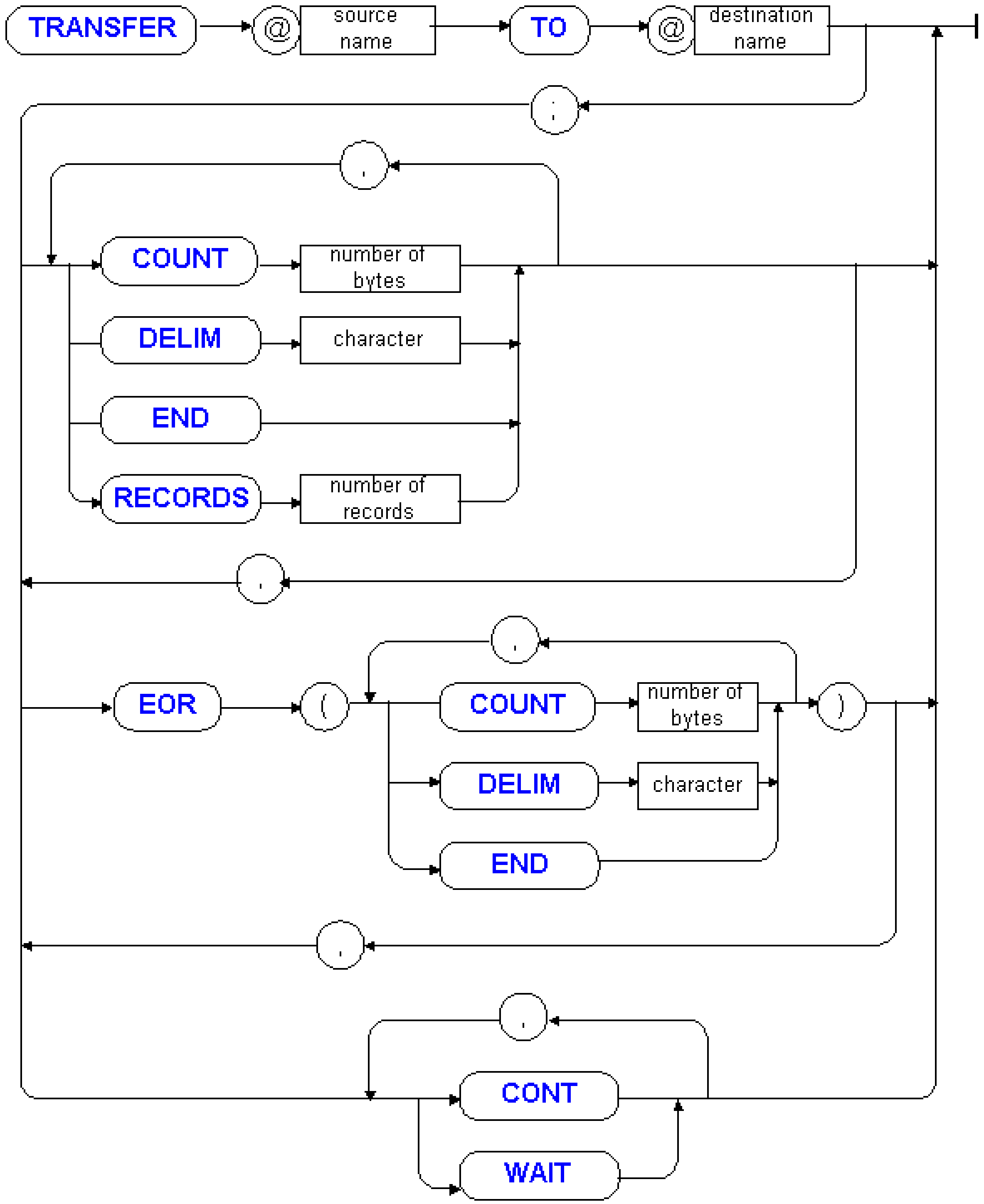
TRANSFER

Performs an unformatted I/O transfer.

Syntax: TRANSFER @source-io-path TO @dest-io-path [; parameters]

where:

parameters = [eot-term-list] [,] [EOR(eor-term-list)] [,] [type]
[,] = the optional comma is only needed when items
occur on both sides of it.
eot-term-list = eot-term [,eot-term...]
eot-term = COUNT bytes |
DELIM character |
END |
RECORDS number
eor-term-list = eor-term [,eor-term...]
eor-term = COUNT bytes | DELIM character | END
type = { CONT | WAIT } [, type]
bytes, number = numeric-expressions, rounded to integers
character = string-expression, zero or one character



Sample: `TRANSFER @Device TO @Buffer`

```
TRANSFER @Buff TO @Logger;CONT
TRANSFER @Rs232 TO @Buff;DELIM CHR$(13)
TRANSFER @Path TO @Buff;RECORDS 16,EOR(END)
```

View Sample: TRANSFER.BAS **(also found in examples directory)**

Description:

The **TRANSFER** statement sets up unformatted data transfers between memory and a device. The data transfer normally occurs in the "background." That is, the BASIC program continues to run in the "foreground" simultaneously with the background transfer. Optionally, the **TRANSFER** statement can wait until the transfer is complete before continuing.

TRANSFER is not supported on all interfaces. The interface hardware must have the necessary circuitry and the device driver must have the proper software support.

Buffers

The transfer operation must be between a buffer and a device. A buffer must be declared as the source for an outbound transfer or as the destination of an inbound transfer. One buffer can simultaneously be used for an outbound transfer and an inbound transfer. A transfer directly between two devices is not supported.

Buffers may be unnamed or named. An unnamed buffer is created, assigned an I/O path and given its size by the ASSIGN statement. A named buffer is a previously declared REAL, INTEGER or COMPLEX array or a string scalar (declared in a COM, DIM, INTEGER, REAL or COMPLEX statement) which has been ASSIGNed to an I/O path. Unnamed buffers are usually preferred because the size can be as large as available memory and no side-effects are possible by accessing the buffer through its variable name.

Buffers are circular; each buffer has a fill and empty pointer as well as a count. The fill pointer is used by an inbound transfer to identify the next location for data to be stored (inserted). The empty pointer is used by an outbound transfer and points to the next location for data to be output (removed). A value of one for either pointer means the first byte of the buffer. When the fill and empty pointers have the same value, the count can be examined to determine whether the buffer is empty or full.

The I/O path assigned to the buffer is called the buffer-I/O path. The I/O path assigned to the device is called the non-buffer-I/O path. The buffer should be accessed only with the buffer-I/O path. The count, fill and empty pointers can be examined using STATUS on the buffer-I/O path. OUTPUT @buf or an inbound transfer are used to place data into a buffer. ENTER @buf or an outbound transfer are used to read and remove data from a buffer. The variable name of a named buffer should generally not be used to access the data in the buffer since the data in the buffer is unformatted and may even have the wrong byte order.

Transfer Type

The type of the transfer can be specified as **CONT**, **WAIT**, or left unspecified.

If **WAIT** is specified, the transfer executes in foreground mode. Program execution does not proceed beyond the **TRANSFER** statement until the transfer terminates. If an error occurs, it is reported with the line number of the **TRANSFER** statement. If **WAIT** is not specified, execution continues past the **TRANSFER**statement and the transfer takes place in the background. Then if an error occurs, the error is not reported until the non-buffer-I/O path is referenced. The error line reported is not that of the **TRANSFER**, but of the statement where the non-buffer-I/O path was referenced.

If **CONT** is specified, **TRANSFER** executes continuously. For an inbound transfer, execution pauses when the buffer is full and continues when space is available in the buffer. For an outbound transfer, execution pauses when the buffer is empty and continues when the buffer has data available. If **CONT** is not specified, the end-of-transfer occurs when an outbound transfer empties the buffer or an in-bound transfer fills the buffer. Or if a termination method has been specified as explained below, the transfer terminates when the condition occurs.

Both **WAIT** and **CONT** can be specified together if a transfer is already active for the buffer in the opposite direction. The transfer will be continuous, but will run in the foreground.

If neither **WAIT** nor **CONT** is specified, the transfer occurs in the background. The end-of-transfer occurs when an outbound transfer empties the buffer or an in-bound transfer fills the buffer. Or if a termination method has been specified as explained below, the transfer terminates when the condition occurs.

Transfer Termination

An *eot-term-list* can be used to specify a list of conditions which cause the transfer to end. The following end-of-transfer termination conditions, *eot-term*, can be used:

If **COUNT** is specified, the transfer terminates after the specified number of bytes has been transferred.

If **DELIM** is specified for an inbound transfer, then the transfer is terminated after the specified character is detected. **DELIM** is not allowed with outbound transfers. If the delimiter string is zero length, delimiter checking is disabled. **DELIM** prevents DMA from being used; interrupts will be used instead.

If **END** is specified for an inbound transfer, the transfer terminates when the device dependent signal is received. On the IEEE-488 interface, **END** is the EOI signal. When an inbound transfer is terminated in this way, bit 3 of register 10 is set. For an outbound transfer, **END** does not specify a termination condition, but rather specifies that the device dependent signal (EOI) is sent with the last byte sent.

If **RECORDS** is specified, the transfer terminates when the specified number of records has been transferred. An *eor-term-list* must be specified, defining what will be considered a record for the purpose of this particular transfer. For inbound transfers the legal end-of-record termination conditions, *eor-term*, are **COUNT**, **DELIM** and **END** or some combination of these three. For outbound transfers only **COUNT** can be used to define a record, although **END** can be used to specify that the device dependent signal (EOI) is sent with the last byte of each record.

The ON EOR and ON EOT statements can be used to generate an event when an end-of-record or end-of-transfer occurs. The WAIT FOR EOR and WAIT FOR EOT statements can be used to stop further statement execution until an end-of-record or end-of-transfer occurs.

To terminate a **CONT**, continuous mode, outbound transfer without leaving data in the buffer, use the following sequence of statements:

```
CONTROL @Buff,8;0
WAIT FOR EOT @Non_buff
```

Hanging and Premature Termination

HTBasic will not enter a stopped state until all transfers are completed. Likewise, HTBasic will not exit a program context until transfers started in that context are finished. The following statements also cause the computer to "hang" until all transfers complete: GET, LOAD, RETURN, STOP, SUBEND, SUBEXIT or modifying a program line.

The ABORTIO statement can be used to prematurely terminate a transfer and free the computer. The RESET key will also terminate any active transfers, but ABORTIO is preferred.

Outbound TRANSFER

An outbound transfer has the form:

```
TRANSFER @Buff TO @Non_buff
```

If another outbound **TRANSFER** statement is executed while an outbound **TRANSFER** is occurring, HTBasic waits for completion of the first before starting the second. Any EOT/EOR events caused by the first transfer will then be logged and may be serviced before the next program line.

Inbound TRANSFER

An inbound transfer has the form:

```
TRANSFER @Non_buff TO @Buff
```

If another inbound **TRANSFER** statement is executed while an inbound **TRANSFER** is occurring, HTBasic waits for completion of the first before starting the second. Any EOT/EOR events caused by the first transfer will then be logged and may be serviced before the next program line.

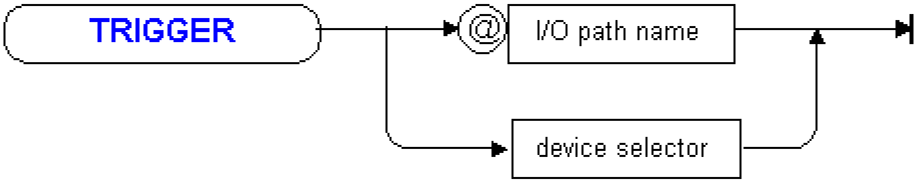
See Also:

ABORTIO, ASSIGN, ENTER, ON EOR, ON EOT, OUTPUT, RESET, STATUS, WAIT

TRIGGER

Sends a trigger message to all or selected devices on the IEEE-488.

Syntax: TRIGGER {@io-path | device-selector}



Sample: `TRIGGER @Gpib`

```
TRIGGER 712
TRIGGER Dev
```

Description:

TRIGGER sends a trigger message to a specified device or to all LISTEN addressed devices on the IEEE-488 bus. The computer must be the active controller. If an I/O path is specified, it must be assigned to the IEEE-488 interface or to one or more IEEE-488 devices.

If primary device addresses are specified bus action is: ATN, UNL, LAG, GET. If only an interface select code is specified the bus action is: ATN, GET.

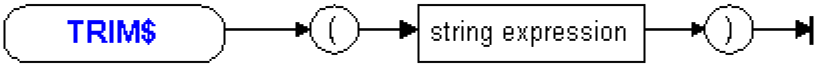
See Also:

ABORT , CLEAR, INTR, LOCAL, PASS CONTROL, PPOLL, REMOTE, REQUEST, SEND, SPOLL

TRIM$

Removes leading and trailing spaces from a string.

Syntax: TRIM$(string-expression)



Sample:

```
A$=TRIM$(B$)
Heading$=TRIM$("       Title  ")
```

View Sample: **TRIM$.BAS** **(also found in examples directory)**

Description:

The **TRIM$** string function removes leading and trailing spaces from a string. The embedded spaces are not affected.

See Also:

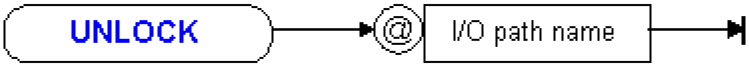
CHR$, LWC$, NUM, REV$, POS, RPT$, UPC$, VAL, VAL$

UNLOCK

Removes exclusive access protection from a LOCKed file.

Syntax: UNLOCK @io-path

where: io-path = name assigned to a file



Sample:

```
UNLOCK @Proprietary
IF Unsecure THEN UNLOCK @File
```

View Sample: **UNLOCK.BAS** **(also found in examples directory)**

Description:

File locking capabilities depend on the operating system HTBasic is running on. If the operating system does not support it, this command is ignored. An ASSIGN @PathTO * will **UNLOCK** and then close the file.

A file can have multiple locks on it. The file remains locked until a corresponding number of **UNLOCK**statements have been executed. LOCKing a file should be a temporary action of short duration so that fair access to the file is provided to all users.

Under DOS or Windows, SHARE may need to be loaded in order to share, lock and unlock files. Consult the manufacturer's documentation for your system. If SHARE is necessary, but not currently installed, the LOCK will fail with an error number 1.

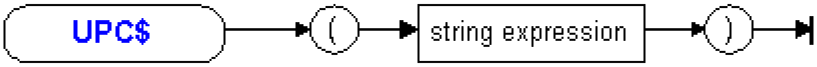
See Also:

ASSIGN, LOCK

UPC$

Converts characters in a string to uppercase characters.

Syntax: UPC$(string-expression)



Sample:
```
A$=UPC$(B$)
Capital$=UPC$(Names$)
```

View Sample: **UPC$.BAS** **(also found in examples directory)**

Description:

LEXICAL ORDER IS determines the lowercase to uppercase correspondence. If the lexical order is a user-defined table and the optional upper and lowercase conversion rules were not specified, the upper to lowercase correspondence is determined by the standard lexical order.

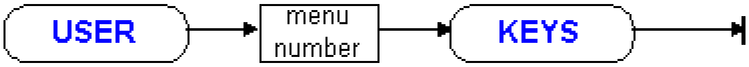
See Also:

CHR$, LWC$, NUM, POS, REV$, RPT$, TRIM$, VAL, VAL$

USER KEYS

Displays the specified User Softkey Menu.

Syntax: USER menu-number KEYS



Sample:
```
USER Menu KEYS
IF Two THEN USER 2 KEYS
```

View Sample: **USER KEYS.BAS** **(also found in examples directory)**

Description:

The menu number may be a numeric expression and is rounded to an integer. It should be in the range one to three.

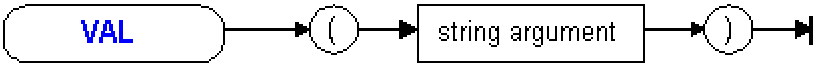
See Also:

KBD CMODE, KEY LABELS, KEY LABELS PEN, SYSTEM KEYS

VAL

Converts a string into a numeric value.

Syntax: VAL(string-expression)



Sample:

```
I=VAL(Response$)
IF VAL(SYSTEM$("VERSION:OS"))<3 THEN CALL Alternate
```

View Sample: **VAL.BAS** **(also found in examples directory)**

Description:

There must be a digit, a plus or minus sign or a decimal point as the first non-blank character of the string. The remaining characters are scanned until a non-numeric character is seen. If an E is present the characters must form a valid number in scientific notation format. **VAL** is the opposite of the VAL$ function.

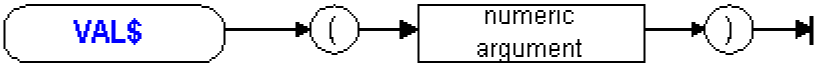
See Also:

DVAL, DVAL$, IVAL, IVAL$, NUM, POS, VAL$

VAL$

Converts a number into its string representation.

Syntax: VAL$(numeric-expression)



Sample: `A$=VAL$(12345)`

```
CREATE "DATA."&VAL$(Version)
```

View Sample: **VAL$.BAS** **(also found in examples directory)**

Description:

The returned string is in default print format, except that no trailing blanks are attached to the string and no leading blank is attached to positive numbers. **VAL$** is the opposite of the VAL function.

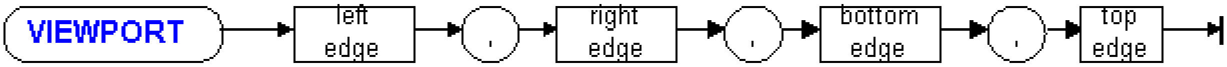
See Also:

DVAL, DVAL$, IVAL, IVAL$, NUM, POS, VAL

VIEWPORT

Defines the area of the graphic device used for output.

Syntax: VIEWPORT left,right,bottom,top



Sample: `VIEWPORT Left,Right,Bottom,Top`

```
VIEWPORT 0,75,10,30
```

View Sample: **VIEWPORT.BAS** **(also found in examples directory)**

Description:

VIEWPORT selects the area of the screen (or device) to be used for graphics output and sets the soft-clip boundary limits. The coordinate system defined by WINDOW or SHOW will be mapped into this area. The left limit must be less than the right limit and the bottom limit must be less than the top limit. The default viewport is the entire surface.

By changing the **VIEWPORT** parameters, you change the proportions, size and position of the drawing surface. Graphic output is automatically scaled to fit this drawing surface. Changing the viewport does not affect any currently displayed graphics, only graphics that you subsequently generate.

Graphic Display Unit Parameters

VIEWPORT soft-clip boundary parameters are defined in GDUs (Graphic Display Units). GDUs are units that describe the physical bounds of the display area on the graphic output device. By definition, Graphic Display Units are 1/100 of the Y axis of a plotting device. A unit in the X direction and the Y direction is of the same length. The RATIO function returns the X to Y hard-clip limits ratio and can be used to determine the **VIEWPORT** soft-clip limits.

The **VIEWPORT** soft-clip limits should not exceed the hard-clip limits. By default the left limit is zero, the right limit is the X axis hard-clip limit, the bottom limit is zero and the top limit is the Y axis hard-clip limit.

Porting Issues

In HTBasic, GDUs are always 100 in the Y direction. In HP BASIC, if the ratio is less than 1, the X axis is 100 GDUs and the Y axis is (100*RATIO) GDUs long; if the ratio is greater than 1, the Y axis is 100 GDUs and the X axis is (100*RATIO) GDUs long.

See Also:

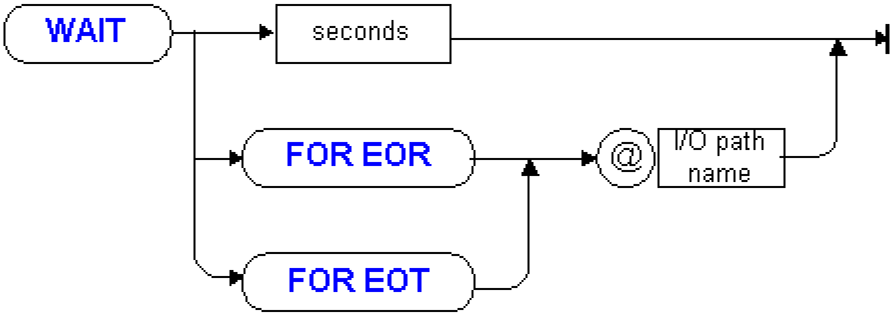
CLIP, RATIO, SHOW, WINDOW

WAIT

Waits a specified time or for TRANSFER events.

Syntax: WAIT seconds
WAIT FOR {EOR|EOT} @io-path

where: seconds = numeric expression



Sample: `WAIT Sec/7`
`WAIT FOR EOR @Device WAIT FOR EOT @Non-buffer`

View Sample: **WAIT.BAS** **(also found in examples directory)**

Description:

If seconds are specified, the computer pauses execution for the length of time specified. The seconds argument must be in the range 0 to 2,147,483.648 seconds. The number is rounded to the nearest millisecond, or to the resolution of the computer clock.

The **WAIT FOR EOR** statement waits until an end-of-record event occurs during a TRANSFER. Similarly, the **WAIT FOR EOT** statement waits until any TRANSFER active on the I/O path is complete. The I/O path must be the I/O path used in the TRANSFER to specify the device. Using the I/O path assigned to the buffer will cause an error. If the I/O path is not involved in an active TRANSFER, the statement has no effect.

See Also:

ON DELAY, PAUSE

WAIT FOR EVENT

Suspends program execution until an event occurs.

Syntax: WAIT timeout value

Sample:

```
WAIT FOR EVENT
WAIT FOR EVENT; TIMEOUT 30
```

Description:

At the WAIT FOR EVENT statement, program execution is suspended until an event occurs. When an enabled event occurs, the WAIT FOR EVENT statement terminates and the event triggers the appropriate pending ON EVENT statement. If no events are currently defined, WAIT FOR EVENT returns immediately.

WAIT FOR EVENT will wait indefinitely for an event to occur unless you use the TIMEOUT option with it. The TIMEOUT option specifies the number of seconds after which program execution resumes if no event has occurred.

The corresponding branch may or may not be taken, depending which has the highest priority: the current context and its priority or the defining context and its priority.

Since the WAIT FOR EVENT statement suspends program execution, the computer is free to service other processes. In the following construct, the computer is “busy waiting” (that is, the CPU stays busy doing nothing):

10 GOTO 10

If keeping the CPU free to run other processes is important in your program or computer environment, we recommend using either of the following two constructs:

```
10   LOOP
20        WAIT FOR EVENT
30   END LOOP
```

or:

```
10   WAIT FOR EVENT
20   GOTO 10
```

NOTE

WAIT FOR EVENT will wait if any events are defined, even if any or all events are disabled or are associated with widgets that are not visible. If the widgets are not visible, WAIT FOR EVENT will terminate only if the timeout period is reached, or if you press the **Stop** or **Reset** keys.

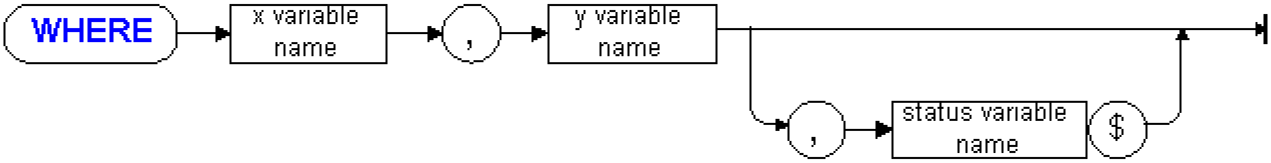
See Also:

ON DELAY, PAUSE

WHERE

Returns the logical pen position.

Syntax: WHERE x-variable, y-variable [,string-name$]



Sample: `WHERE X,Y`

```
WHERE Time,Temp,Status$
```

View Sample: WHERE.BAS **(also found in examples directory)**

Description:

The **WHERE** statement returns the current logical pen position in the x and y numeric variables and pen status information in the optional string variable.

The optional string variable must be dimensioned to a length of at least three bytes. The three string characters are interpreted as follows:

Byte	**Meaning**
1	Pen Status - Up/Down status of the Pen. If the character is a "1" then pen is down; if it is a "0" then the pen is up.
2	Comma delimiter character.
3	Clip Indicator - If the character is a "0", then the point is outside the P1, P2 limits. If a "1", the point is inside the P1, P2 limits, but outside the viewport. If a "2" then it's inside the viewport.

See Also:

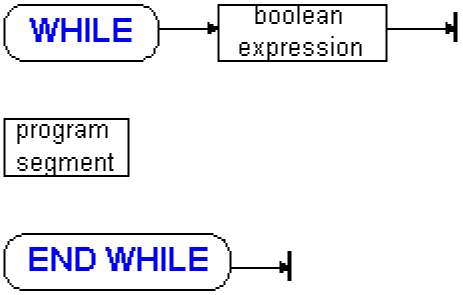
DIGITIZE, GRAPHICS INPUT IS, PLOTTER IS, READ LOCATOR, SET ECHO, SET LOCATOR, TRACK

WHILE

Repeats an action while a condition is true.

Syntax: WHILE numeric-expression
statements
END WHILE

where: statements = zero, one or more program statements



Sample:

```
100 WHILE X<1000
. . .
200 END WHILE
```

View Sample: **WHILE.BAS** **(also found in examples directory)**

Description:

The **WHILE** expression is evaluated and if false (zero), execution continues with the statement following the **END WHILE**. If true (non-zero), then the statements in the **WHILE** loop are executed. When the **END WHILE** is reached, execution branches back to the **WHILE** statement where the expression is again evaluated.

See Also:

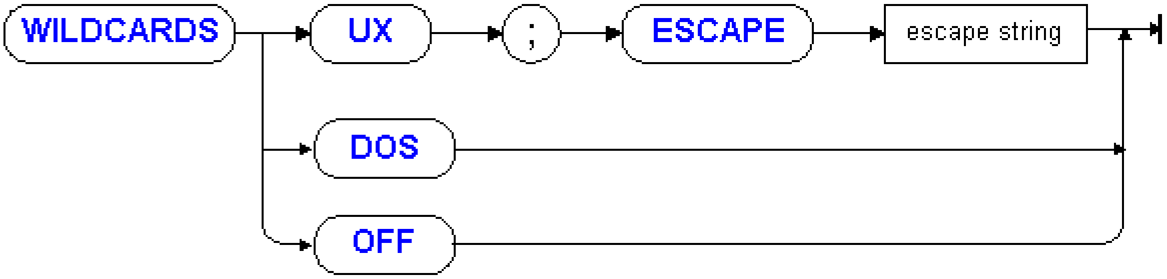
FOR, LOOP, REPEAT

WILDCARDS

Enables or disables wildcard support.

Syntax: WILDCARDS [OFF | DOS | UX; ESCAPE char]

where: char = string expression evaluating to "\", "'" or ""



Sample:

```
WILDCARDS OFF
WILDCARDS DOS
WILDCARDS UX;ESCAPE "\"
```

View Sample: **WILDCARDS.BAS** **(also found in examples directory)**

Description:

Wildcards are characters which can be used in a filename as a template to select a group of files to be operated upon. A filename with wildcard characters in it will be compared with existing filenames using special rules and all filenames that "match" are acted upon. It is necessary in HP BASIC/WS to support wildcards in many commands since no operating system is available. Under HTBasic, wildcards can be used directly in operating system commands using the EXECUTE statement. However for convenience, wildcards are supported in the CAT statement.

The question mark "?" and the asterisk "*" are the wildcard characters. If the **WILDCARDS** statement is executed it will return an error because wildcarding is always on. SYSTEM$ ("WILDCARDS") always returns "ON:". Wildcarding never needs to be turned off because the wildcard characters are not legal filename characters.

These are the rules used to match an actual filename with wildcards:

1. The "?" character will match any one character in the same position of an actual filename. For example, the string "?AT" will match the strings "CAT", "BAT", "MAT" or any other string three letters long which has an "A" as the second letter and "T" as the third letter.

2. The "*" character will match zero or more characters. For example, "*" will match all filenames. "F*" will match all filenames starting with the letter "F". "*.BAS" will match all filenames which have the ".BAS" extension.

FAT file systems with long file names exhibit an unexpected behavior. If the wildcards match either the 8.3 name or the long name, the file is considered to match. The state of CONFIGURE LONGFILENAMES has no effect.

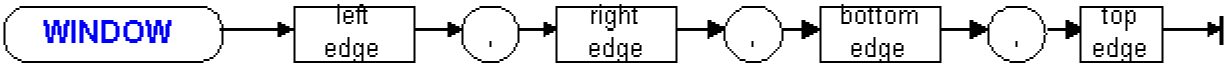
See Also:

CAT, SYSTEM$

WINDOW

Sets the bounds for displayable graphics data in user defined units.

Syntax: WINDOW left,right,bottom,top



Sample:

```
WINDOW 0,X,-100,100*X*RATIO
WINDOW -10,10,0,50
WINDOW 10,-10,50,0
```

Description:

WINDOW defines the units to be displayed within the VIEWPORT or the hard-clip boundaries. They can be any units of measure you wish to work with (inches, miles, years, etc.). The default **WINDOW** setting is equal to the default VIEWPORT setting.

The **WINDOW** and SHOW statements differ in how they map data onto the viewport. **WINDOW** may use non-isotropic units (the X and Y units are of different lengths); whereas SHOW uses isotropic units (the X and Y units are of equal length).

An image can be "mirrored" about the X or Y axes by reversing the order of the limits for each dimension by specifying the high value before the low value.

See Also:

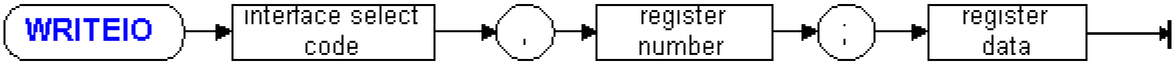
CLIP, SHOW, VIEWPORT

WRITEIO

Writes to a hardware register or a memory byte/word.

Syntax: WRITEIO interface-select-code, hardware-register; data
WRITEIO special-interface, address; data

where: hardware-register, data = numeric-expressions rounded to integers
special-interface = numeric-expression rounded to integer, legal values are explained in the description
address = numeric-expression rounded to a linear address



Sample:

```
WRITEIO Centronix,0;&HAA
WRITEIO -9826,Address;New_value
WRITEIO 8080,Ioadd;BINIOR(Oldata,&H80)
```

Description:

Hardware Registers

The specified data value is written to a hardware interface register. READIO/**WRITEIO** operations should not be mixed with STATUS/CONTROL operations. Do not attempt to use READIO/**WRITEIO** registers unless you are very familiar with the hardware. Use the STATUS/CONTROL registers instead. The hardware manuals for your computer should be consulted for complete documentation on the interface hardware. The *User's Guide* lists READIO/**WRITEIO** registers for the interface device drivers included with HTBasic. Optional interface device drivers include documentation for the registers.

READIO/**WRITEIO** registers in HTBasic are not compatible with HP BASIC READIO/**WRITEIO** registers when the interface hardware is not the same. TransEra's IEEE-488 card uses the same IEEE-488 chip as HP's HP-IB, therefore the READIO/**WRITEIO** registers are identical. The serial interface hardware registers differ not only if the UART chip is different, but also if the circuitry surrounding the chip is different. The TransEra GPIO interface is READIO/**WRITEIO** compatible with HP's GPIO.

Special Interface Select Codes

There are a number of special interface select codes which can be used with the **WRITEIO** statement. The legal values for special-interface are given in the following paragraphs. For compatibility with earlier releases of HTBasic, **WRITEIO** 8080,L and **WRITEIO** -8080,L are still supported but should be replaced with OUT and OUTW, respectively.

POKE Memory

WRITEIO 9826,L;V and **WRITEIO** -9826,L;V are used to "poke" the value V into a byte or word of memory, respectively. L specifies the address of the byte/word to poke. If L is odd when doing a word operation, the even address L-1 is used. L specifies an address within the HTBasic process.

Warning: Poke should only be done on addresses returned by READIO(9827,I)! Poking any other location can cause your system to crash, data to be lost and damage to your computer hardware. Use of this function for any other address is unsupported, and TransEra cannot be held responsible for any consequences. Under some protected mode operating systems like Windows NT, some of these operations are not allowed.

See Also:

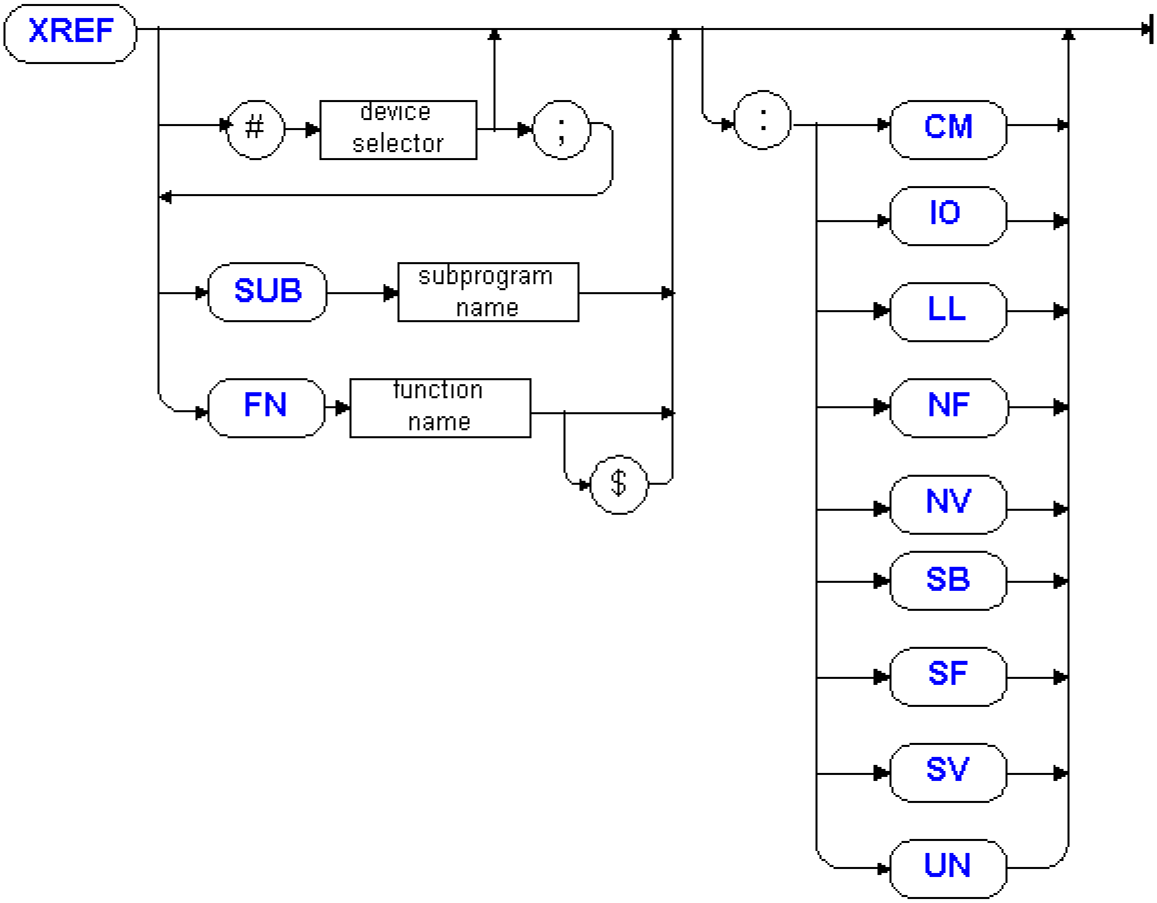
CONTROL, INP, OUT, READIO, STATUS

XREF

Generates a cross reference of a program.

Syntax: XREF [[SUB] sub-name] [: option]
XREF [#device-selector [; [SUB] sub-name]] [: option]

where: sub-name = subprogram-name | FN function-name[$] | string-expression
option = CM | IO | LL | LN | NF | NV | SB | SF | SV| UN



Sample: `XREF`

```
XREF Trigger:NV
XREF #701;Launch
```

Description:

XREF generates a cross reference list of line labels and numbers, io-path names, numeric and string variables, subprograms, functions and COM block names. It also lists the number of unused symbol table entries. The listing is sent to the PRINTER IS device unless a device selector is specified.

Optional parameters include:

Option	**Meaning**
CM	Common Block Names
IO	I/O Path Names
LL	Line Labels
LN	Line Numbers
NF	Numeric Functions
NV	Numeric Variables
SB	SUB Subprograms
SF	String Functions
SV	String Variables
UN	Unused Entries

If a reference is a SUB parameter, declared in a COM, COMPLEX, DIM, REAL or INTEGER statement or a line label, it is marked by the "<-DEF" marker. After each program context, the number of unused symbol table entries is displayed. If the subprogram name is specified as MAIN, the MAIN context is cross-referenced.

If the program has been pre-run, array variables will be noted as Array next to the defination line number.

Subprogram Pointer

If a string expression specify the subprogram name in the **XREF** statement, the string expression is called a subprogram pointer because it "points" to the subprogram rather than explicitly naming it. As the expression changes, the pointer points to different subprograms. The subprogram must be specified with the initial character in uppercase, and subsequent characters in lowercase. Subprogram pointers can also be used in CALL, DELSUB, INMEM, and LOADSUB statements.

Porting to HP BASIC:

The use of subprogram pointers in **XREF** is a new HTBasic feature that is not available in HP BASIC. It should not be used in programs that must be ported back to HP BASIC.

See Also:

PRINTALL IS, TRACE

Error Codes
1 to 25

Error 1	Missing Option or Configuration Error.
Error 2	Memory Overflow.
Error 3	Line not Found in Current Context.
Error 4	Improper RETURN.
Error 5	Improper Context Terminator.
Error 6	Improper FOR/NEXT Matching.
Error 7	Undefined Function or Subprogram.
Error 8	Improper Parameter Matching.
Error 9	Improper Number of Parameters.
Error 10	String Type Required.
Error 11	Numeric Type Required.
Error 12	Attempt to Re-declare Variable.
Error 13	Array Dimensions not Specified.
Error 14	OPTION BASE not allowed here.
Error 15	Invalid bounds.
Error 16	Improper or Inconsistent Dimensions.
Error 17	Subscript out of Range.
Error 18	String Overflow or Sub-string Error.
Error 19	Improper Value or out of Range.
Error 20	INTEGER overflow.
Error 22	REAL overflow.
Error 24	Trig argument too large.
Error 25	Magnitude of ASN or ACS >> 1.0

Error Codes 26 to 49

Error 26	Zero to negative power.
Error 27	Negative base to non-integer power.
Error 28	LOG or LGT of a non-positive number.
Error 29	Illegal floating point number.
Error 30	SQR/SQRT of a negative number.
Error 31	Division (or MOD) by zero.
Error 32	String is not a valid number.
Error 33	Improper arg for NUM or RPT$.
Error 34	Line not an IMAGE Statement.
Error 35	Improper IMAGE Statement.
Error 36	Out of data in READ.
Error 38	TAB or TABXY not allowed here.
Error 40	Improper COPYLINES, MOVELINES or renumber.
Error 41	First line number greater than second.
Error 43	Non-square Matrix.
Error 44	Result cannot be an operand.
Error 46	No program in memory.
Error 47	Incorrect or inconsistent COM declarations.
Error 49	Branch destination not Found.

Error Codes 50 to 99

Error 51	File not currently Assigned.
Error 52	Improper MSUS.
Error 53	Improper File Name.
Error 54	Duplicate File Name.
Error 55	Directory Overflow.
Error 56	File or Path not found.
Error 58	Improper File Type.
Error 59	End of File or Buffer.
Error 60	End of Record.
Error 64	Mass Storage Media Overflow.
Error 65	Incorrect Data Type.
Error 67	Illegal Mass Storage Parameter.
Error 68	Syntax Error during GET.
Error 72	Drive Not Found.
Error 80	Disk changed or not in Drive.
Error 82	Mass Storage unit not present.
Error 83	Write Protected.
Error 84	Sector not Found.
Error 85	Media not Initialized.
Error 88	READ Data Error.
Error 89	Checkread error.
Error 90	Mass storage system error.

Error Codes
100 to 149

Error 100	Numeric IMAGE field for String Item.
Error 101	String IMAGE field for Numeric Item.
Error 102	Numeric Field specifier is too Large.
Error 103	Data item has no corresponding IMAGE specifier.
Error 105	Numeric Field specifier is too Small.
Error 106	IMAGE exponent field too Small.
Error 107	IMAGE sign specifier missing.
Error 117	Too many nested structures.
Error 118	Too many structures in context.
Error 120	Not allowed while program running.
Error 122	Program is not Continuable.
Error 128	Line too long during GET or a CHANGE.
Error 131	Unrecognized Keycode.
Error 133	DELSUB of non-existent or busy subprogram.
Error 134	Improper Scratch Statement
Error 136	REAL underflow.
Error 141	Variable already allocated.
Error 142	Variable not Allocated.
Error 143	Reference to missing OPTIONAL Parameter.
Error 145	May not build COM at this time.
Error 146	Duplicate Line label in this Context.

Error Codes 150 to 299

Error 150	Bad select code or device specifier.
Error 153	Insufficient data for ENTER.
Error 155	Improper Interface Register number.
Error 157	No ENTER terminator found.
Error 158	Improper IMAGE specifier or nesting.
Error 159	Numeric data not received.
Error 163	Interface not present.
Error 164	Illegal BYTE/WORD operation.
Error 167	Interface Status Error.
Error 168	Device Timeout.
Error 170	I/O operation not allowed.
Error 171	Illegal I/O addressing sequence.
Error 172	Peripheral Error.
Error 173	Active or System Controller Required.
Error 177	Undefined I/O Path Name.
Error 183	Permission denied.
Error 186	Cannot open the specified directory.
Error 187	Cannot link across devices.
Error 188	Cannot rename with "." or "..".
Error 189	Too many open files.
Error 190	File size too big.
Error 191	Too many links to a file.
Error 193	Resource deadlock would occur.
Error 194	Operation would block.
Error 195	Too many levels of symbolic link.
Error 196	Target device busy.
Error 290	Invalid ESCAPE character

Error Codes
300 to 459

Error 330	LEXICAL ORDER IS array too small.
Error 331	Repeated subscript in REORDER vector.
Error 332	Non-existent dimension given.
Error 333	Improper subscript in REORDER vector.
Error 334	REORDER vector has wrong size.
Error 335	Indirection array is not a Vector.
Error 338	Key subscript out-of-range.
Error 340	Table Length Error.
Error 341	Order Table Lower Byte Error.
Error 342	Not a One-dimensional INTEGER Array.
Error 343	Special Case Index is Too Big.
Error 344	2-to-1 List Length Error.
Error 346	INDENT parameter out of range.
Error 347	Structures improperly matched.
Error 401	Bad system function argument.
Error 427	Priority may not be lowered.
Error 435	EXEC not allowed on this Binary.
Error 453	File in Use.
Error 455	Possibly corrupt file.
Error 456	Unsupported directory operation.
Error 459	Specified file is not a directory.

Error Codes 460 to 699

Error 460	Directory not empty.
Error 462	Invalid Password.
Error 465	Invalid rename across volumes.
Error 471	TRANSFER not supported by Interface.
Error 481	File locked or open Exclusively.
Error 482	Not allowed with a directory.
Error 485	Invalid Volume Copy.
Error 511	MAT INV result array must be REAL.
Error 543	Improper Dimensions for REDIM.
Error 553	Cannot load object file.
Error 602	Improper BUFFER Lifetime.
Error 603	Variable not declared BUFFER.
Error 604	Bad TRANSFER source or destination.
Error 606	Improper TRANSFER parameters.
Error 609	IVAL/DVAL result too large.
Error 611	Premature TRANSFER termination.
Error 612	BUFFER pointers in use.
Error 620	Complex value not allowed here.
Error 623	ATN is undefined at +/- i.
Error 624	ACSH/ATNH argument out of range.
Error 625	Bad SEARCH condition on Complex.

Error Codes 700 to 899

Error 700	Improper Plotter specifier.
Error 704	Upper bound not greater than lower bound.
Error 705	VIEWPORT/CLIP Beyond Hard Clip Limits.
Error 708	Device not initialized.
Error 713	Request not supported by device.
Error 730	Internal error occurred in library call.
Error 733	GESCAPE opcode not recognized.
Error 810	Feature not supported on this system.
Error 815	Cannot access system time.
Error 826	EXECUTE process status failure.
Error 827	String too long for EXECUTE.
Error 831	Write to a broken pipe.
Error 832	Cannot seek on a pipe.
Error 833	Wrong direction data transfer in pipe.
Error 841	CSUB run-time error.
Error 863	Not in a window system.
Error 898	Softkey Macro is too long.
Error 899	Key number out of range.

Error Codes 900 to 999

Error 900	Undefined softkey macro.
Error 901	Softkey Macro memory overflow.
Error 902	Must delete entire context.
Error 903	No line number room to renumber.
Error 905	CHANGEd line too long.
Error 906	SUB or DEF FN not allowed here.
Error 909	May not replace SUB or DEF FN.
Error 910	Identifier not found in context.
Error 935	Identifier too long.
Error 936	Unrecognized Character.
Error 937	Invalid OPTION BASE.
Error 940	Duplicate formal parameter name.
Error 949	Syntax error at cursor.
Error 951	Incomplete Statement or Command.
Error 956	Source/destination mismatch.
Error 962	Programmable only.
Error 963	Command only.
Error 977	Statement or Command too complex.
Error 980	Too many symbols in context.
Error 985	Invalid Quoted String.
Error 987	Invalid Line Number.

Error Codes 1000-2099

Error 1100	Unable to load DLL.
Error 1101	Unable to unload DLL.
Error 1102	DLL is already loaded.
Error 1103	Unable to load function.
Error 1104	Function is already loaded.
Error 1105	Alias previously used.
Error 1106	Invalid or no return type specified.
Error 1107	Invalid or no DLL name specified.
Error 1108	Insufficient Dimension for passing string by reference.
Error 1109	Unsupported Number of Parameters.
Error 1110	Possible HTBasic Memory Corruption
Error 2000	Stack Overflow.
Error 2001	Too many Open Files.
Error 2002	HELP file not found.
Error 2003	Bad Device Driver number.
Error 2004	Bad Key Function number.
Error 2005	Illegal in Run-only Version.
Error 2006	Illegal DUMP device.
Error 2007	Wrong Object Type.
Error 2008	May not modify CSUB.
Error 2009	Wrong Revision.
Error 2010	May Not load driver Here.
Error 2011	Exceeded Graphics Driver Limit.
Error 2012	Illegal CALL in CSUB.

Error Codes 10,000+

Errors over 10,000 are errors passed through HTBasic from the Windows Operating System.

Error 1
Missing Option or Configuration Error

The operation you were attempting is not available in this version. Because of the limitations of some computer systems, not all statements and functions are available in every version of HTBasic. When porting HP BASIC programs to HTBasic, if this error occurs, check the *Reference Manual* entry for more information.

Error 2
Memory Overflow

There is not enough free memory for the requested operation. The -w switch, explained in the *Installing and Using* manual, may solve the problem.

Error 3
Line not Found in Current Context

The specified program line could not be found in this context.

Error 4
Improper RETURN

A RETURN or ERROR RETURN was executed while not inside a subroutine or a user defined function.

Error 5
Improper Context Terminator

No END statement was found for the MAIN context, SUBEND statement for a subprogram or RETURN and FNEND statements for a user defined function.

Error 6
Improper FOR/NEXT Matching

Either FOR...NEXT loops overlap or a FOR or NEXT statement is missing.

Error 7
Undefined Function or Subprogram

The specified user defined function or subprogram is not currently in memory or could not be found in the file.

Error 8
Improper Parameter Matching

The data type of an argument in a CALL/FN did not match the data type of the associated parameter in the SUB/DEF FN statement.

Error 9
Improper Number of Parameters

There are either too many or too few parameters in the CALL or FNxxx statement.

Error 10
String Type Required

A numeric value was specified in a place where a string value is required.

Error 11
Numeric Type Required

A string value was specified in a place where a numeric value is required.

Error 12
Attempt to Redeclare Variable

The variable has already appeared in an ALLOCATE, DIM, REAL, INTEGER, COM, SUB or DEF FN statement and cannot be redeclared.

Error 13
Array Dimensions not Specified

An attempt was made to use an array which is not dimensioned. Press the PRT ALL key and try the operation again to see the names of all arrays in the program which are not dimensioned.

Error 14
OPTION BASE not allowed here

A DIM, REAL, INTEGER, COM or OPTION BASE statement has already been processed. The OPTION BASE statement must appear before any of these statements. Only one OPTION BASE is allowed per context.

Error 15
Invalid bounds

The array bounds specified are not valid. The lower bound must be less than the upper bound. Each bound must be between -32768 and 32767. The size of a dimension cannot be larger than 32767.

Error 16
Improper or Inconsistent Dimensions

Several conditions return this error: The number of subscripts specified conflicts with the RANK of the array. The size of a dimension cannot be larger than 32767. The dimension specified in a function such as BASE is less than one or greater than the RANK of the array. This array has not been declared. The number of dimensions or elements in this array are not proper for the attempted operation.

If CONFIGURE DIM is OFF, this error also occurs if the variable has not been declared.

Error 17
Subscript out of Range

A subscript value is outside the specified dimension bounds.

Error 18
String Overflow or Sub-string Error

The string value is either too long to fit or the sub-string is incorrectly specified. An overflow can occur when a string becomes longer than 32767, longer than the declared length of the variable it is assigned to, or when a string becomes too long for the internal buffers used in an operation.

Error 19
Improper Value or out of Range

The specified value is not within the valid range. Consult the "Keyword Dictionary" chapter for this operation to find the valid range of values.

Error 20
INTEGER overflow

The value calculated exceeds the range that an INTEGER variable can hold: -32768 through +32767.

Error 22
REAL overflow

The value calculated is too big to be represented by the REAL data type. See MINREAL and MAXREAL in the "Keyword Dictionary" chapter.

Error 24
Trig argument too large

If the argument to a trigonometric function gets too large, it can not be evaluated correctly. If you get this error, you may wish to examine your algorithm or use range reduction.

Error 25
Magnitude of ASN or ACS > 1

The argument to the ASN and ACS functions must be less than one.

Error 26
Zero to negative power

The number zero can only be raised to positive powers or to the zeroth power.

Error 27
Negative base to non-integer power

An attempt was made to raise a negative number to a fractional power.

Error 28
LOG or LGT of a non-positive number

The argument to the LOG and LGT functions can not be negative or zero.

Error 29
Illegal floating point number

The number encountered was not a valid REAL number.

Error 30
SQR/SQRT of a negative number

You cannot take the square root of a negative number.

Error 31
Division (or MOD) by zero

The divisor specified was zero or an operation was attempted that resulted in a division by zero (for example, SHOW 1,1,1,1).

Error 32
String is not a valid number

The characters in the string do not represent a valid numeric value.

Error 33
Improper arg for NUM or RPT$

The resultant string must be less than 32767 characters in length, and the original string must be greater than 0 characters in length.

Error 34
Line not an IMAGE Statement

The program line specified for the USING image was not an IMAGE statement.

Error 35
Improper IMAGE Statement

The IMAGE string or statement is zero length.

Error 36
Out of data in READ

There are no DATA statements that have not been read. Use the RESTORE statement if you wish to re-read existing DATA statements.

Error 38
TAB or TABXY not allowed here

The tab functions are not allowed in this statement.

Error 40
Improper COPYLINES, MOVELINES or renumber

The line numbers specified cannot be used for this operation because: the program sections overlap, line number is not in the range 1 to 4,194,304, the renumber increment is zero, there is not enough room to renumber or a SUB/DEF statement is included, or the destination is not the last program line.

Error 41
First line number greater than second

In a line number range the first line number must be smaller than the second.

Error 43
Non-square Matrix

The array specified does not have the same dimension size in the first and second dimensions, i.e., it is not "square."

Error 44
Result cannot be an operand

The result matrix is not allowed to be one of the operand matrices.

Error 46
No program in memory

There are no program lines in memory or in the range specified.

Error 47
Incorrect or inconsistent COM declarations

The COM statement specifies either a different number of variables or different dimensions than a previous COM statement specified.

Error 49
Branch destination not Found

The ON statement branch destination specified is not defined.

Error 51
File not currently Assigned

The I/O path involved in this operation must be ASSIGNed to a file.

Error 52
Improper MSUS

The Path Specifier (formerly Mass Storage Unit Specifier) is invalid.

Error 53
Improper File Name

The file name specified contains illegal characters or is not of the proper format for this operating system.

Error 54
Duplicate File Name

A file, directory or device, already exists with this name. If you are trying to save a program, use the RE-SAVE or RE-STORE statements to overwrite the existing file. Use the PURGE statement to remove the file.

Error 55
Directory Overflow

The specified mass storage device directory is full. You must either remove an existing file, PURGE or change the size of the directory.

Error 56
File or Path not found

No file or directory exists with this name. You may have forgotten to include the proper device or path specifiers.
Use CREATE or CREATE DIR if you wish to create a new file or directory with this name.

Error 58
Improper File Type

The file type is incorrect for the requested operation or an attempt was made to LOAD an old revision PROG file.

A widget create attempt tried to load a non-widget file.

Error 59
End of File or Buffer

The end-of-file or end-of-buffer was reached unexpectedly during this operation.

Error 60
End of Record

The end-of-record was reached unexpectedly during a random file operation. Either the record size specified in the CREATE BDAT was too small, or the program is attempting to write too much into one record.

Error 64
Mass Storage Media Overflow

The mass storage device is full. This error is also returned when accessing a device through its operating system name (rather than an interface select code) and the device refuses to accept output for any reason.

Error 65
Incorrect Data Type

The array data type is incorrect for this operation. Consult the "Keyword Dictionary" chapter to see if the required type is INTEGER, REAL, or string. Some versions of HTBasic require specific data formats for full-screen GLOADs. Refer to the *Installing and Using* manual.

Error 67
Illegal Mass Storage Parameter

A mass storage parameter, such as the record number, was illegal. Record numbers start at one, not zero.

Error 68
Syntax Error during GET

At least one of the incoming program lines has invalid syntax.

Error 72
Drive Not Found

The specified drive was not found. You must either specify a drive which is legal for your operating system or specify an HP style volume and define a translation for it using the CONFIGURE MSI statement.

Error 80
Disk changed or not in Drive

The disk drive is not ready. The disk drive door may be open or a disk has just been inserted and the drive is not yet ready.

Error 82
Mass Storage unit not present

The specified device is not available. Specifying a non-existent device can cause this error. The unit number is unknown.

Error 83
Write Protected

The disk, device, directory or file is write protected.

Error 84
Sector not Found

The disk may have been initialized in a non-standard way. If an attempt is made to use an HP LIF disk, this error will be returned in most cases, since the disk format is different. You must use disks which have been formatted (initialized) for your operating system.

Error 85
Media not Initialized

The disk drive was not able to find any format information on the disk. The disk has not been initialized or it was initialized on a system whose disk format is alien to your operating system. A "General Failure" reported by a device driver will also cause this error.

Error 88
READ Data Error

The disk controller reported a READ error. This is usually caused by physical or magnetic damage to the data recorded on the disk.

Error 89
Checkread error

A verify check of the data on the disk failed. The disk may be physically or magnetically damaged.

Error 90
Mass storage system error

The operating system reported that it could not complete the requested operation.

Error 100
Numeric IMAGE field for String Item

For example, PRINT USING "D";S$.

Error 101
String IMAGE field for Numeric Item

For example, PRINT USING "A";X.

Error 102
Numeric Field specifier is too Large

The resulting number would be too long for the internal buffers to handle.

Error 103
Data item has no corresponding IMAGE specifier

For example, PRINT USING "X";PI.

Error 105
Numeric Field specifier is too Small

The number will not fit in the specified field width. For example, PRINT USING "D";12.

Error 106
IMAGE exponent field too Small

The exponent value will not fit in the specified field width. For example, PRINT USING "3DEE";1E200.

Error 107
IMAGE sign specifier missing

A negative data item corresponds to an IMAGE specifier that does not include a sign specifier. For example, PRINT USING "D";-1.

Error 117
Too many nested structures

There are too many nested program structures in the program.

Error 118
Too many structures in context

There are too many FOR/NEXT loops in the program context.

Error 120
Not allowed while program running

FIND, CHANGE, COPYLINES, MOVELINES, REN, RUN, CONT, SCRATCH, EDIT, and adding, deleting or changing a program line are not allowed while a program is running.

Error 122
Program is not Continuable

The program must be paused to be able to continue running.

Error 128
Line too long during GET or a CHANGE

Program lines are limited to 256 characters in a LIF ASCII input file or the result of a CHANGE makes the program line longer than 256 characters.

Error 131
Unrecognized Keycode

The specified keycode is not valid. The key pressed has not been assigned to a function or keycodes OUTPUT to the KBD device were illegal.

Error 133
DELSUB of non-existent or busy subprogram

The specified subprogram either does not exist in memory, has been called or is specified in an active ON statement.

Error 134
Improper Scratch Statement

The second keyword was not A, ALL, B, BIN, C, COM, KEY, R or RECALL.

Error 136
REAL underflow

The value specified or calculated is too small to be represented by the REAL data type. MINREAL is the smallest absolute value representable by the REAL data type.

Error 141
Variable already allocated

This variable has already been ALLOCATEd and cannot be ALLOCATEd again until it is first DEALLOCATEd.

Error 142
Variable not Allocated

This variable has not been allocated memory space. An ALLOCATE statement must be executed before this operation can be done.

Error 143
Reference to missing OPTIONAL Parameter

The CALL to the subprogram or function did not specify an argument for this parameter.

Error 145
May not build COM at this time

New COM blocks may not be built during a LOADSUB but must be specified in the MAIN context or a subprogram when the program is first run.

Error 146
Duplicate Line label in this Context

Two line labels have the same name in a context. Make one a different name.

Error 150
Bad select code or device specifier

The interface select code or device specifier is invalid.

Error 153
Insufficient data for ENTER

Not enough values were found in the input data before a terminator was found.

Error 155
Improper Interface Register number

This register number is not supported by this interface or I/O path.

Error 157
No ENTER terminator found

The proper termination was not received during the ENTER. Depending on the operation, terminators might be the line-feed character or the EOI signal. ENTER USING can be used to accept data from sources which do not use the default terminators.

Error 158
Improper IMAGE specifier or nesting

The IMAGE specifier is either invalid or incorrectly nested. See IMAGE in the "Keyword Dictionary" chapter for the correct syntax.

Error 159
Numeric data not received

No numeric value was found in the input data. Make sure that the device is sending ASCII digits before it sends an EOI.

Error 163
Interface not present

There is no interface with the interface select code specified. For some interfaces, a driver must be loaded with the LOAD BIN statement before the interface is available to HTBasic. Consult the *Installing and Using* manual for more information.

Error 164
Illegal BYTE/WORD operation

The specified operation is not allowed for a BYTE or WORD value.

Error 167
Interface Status Error

An error condition has occurred on the interface, such as a UART error on a serial interface.

Error 168
Device Timeout

The device did not respond to the I/O operation within the timeout specified.

Error 170
I/O operation not allowed

An attempt was made to do an illegal operation. The following are some problems to consider. The device may not support the operation. Or a primary address was specified and shouldn't be. Or the operation requires the controller to be or not be active/system controller. USING is not allowed with a LIF ASCII file. For more information, check the "Keyword Dictionary" chapter for the statement being executed and check the documentation for the device driver being accessed.

Error 171
Illegal I/O addressing sequence

IEEE-488 talk, listen and secondary addresses must be in the range 0 to 31. DAQ secondary addresses must be set. DAQ secondary addresses must be set.

Error 172
Peripheral Error

A hardware error occurred. Refer to the driver documentation for more information.

Error 173
Active or System Controller Required

The system must be the active or system controller for this operation.

Error 177
Undefined I/O Path Name

The I/O path name has not been ASSIGNed to a device, file or buffer.

Error 183
Permission denied

You do not have the correct permissions for the operation attempted. Common problems are: Search permission is denied for a component of the path. You do not have read/write permission for the file specified or for the directory the file/directory exists in. The first part of the file is locked so an ASSIGN statement can't complete.

Error 186
Cannot open the specified directory

An error was returned by the operating system when one of the specified directories was accessed.

Error 187
Cannot link across devices

The operating system requires that this type of LINK refer to a file that is on the same mass storage device. If you have multiple devices and are not sure where they are mounted in the directory tree, ask your system administrator.

Error 188
Cannot rename with "." or ".."

An attempt was made to rename "." or "..". These names are fixed and can not be renamed.

Error 189
Too many open files

The limit to the number of simultaneously open files has been reached. DOS allows this number to be changed with the FILES=xxx line in the CONFIG.SYS boot file, however no normal DOS process may have more than 20 open files. Error number 2001 used to be returned by HTBasic for this condition. Now that HP BASIC has added this error, HTBasic has been changed for compatibility.

Error 190
File size too big

The operating system has a maximum limit to the size of a file and that limit has been exceeded.

Error 191
Too many links to a file

The link count of the file/directory would exceed the maximum allowed.

Error 193
Resource deadlock would occur

An attempt was made to lock a system resource that would have resulted in a deadlock situation.

Error 194
Operation would block

The device is in use. Attempting this operation at this time would suspend HTBasic.

Error 195
Too many levels of symbolic link

Too many symbolic links were encountered in translating the pathname specified.

Error 196
Target device busy

The file/directory could not be deleted or renamed because it is the mount point for a mounted file system, is being used by another process, or is the current directory, ".".

Error 290
Invalid ESCAPE character

The set of valid wildcard escape characters is explained in the "Keyword Dictionary" chapter entry for WILDCARDS.

Error 330
LEXICAL ORDER IS array too small

The array specified in the LEXICAL ORDER statement must have at least 257 elements. If the length specified in the 257th element is not zero, there must be that many more elements in the array. Remember the OPTION BASE when figuring the number of elements.

Error 331
Repeated subscript in REORDER vector

The "MAT REORDER..BY X,D" statement requires that the subscripts specified in X be unique.

Error 332
Non-existent dimension given

The dimension specified in a BASE, SIZE or MAT REORDER statement is less than one or greater than the RANK of the array.

Error 333
Improper subscript in REORDER vector

The "MAT REORDER..BY X,Dim" statement requires that the subscripts specified in X be legal subscripts for the specified dimension (i.e., in range BASE(Dim) to BASE(Dim)+SIZE(Dim)-1).

Error 334
REORDER vector has wrong size

The MAT REORDER..BY X statement requires that the SIZE of X be the same as the SIZE of the array dimension being acted upon.

Error 335
Indirection array is not a Vector

The MAT REORDER..BY X and MAT SORT...TO X statements require that X be a vector.

Error 338
Key subscript out-of-range

In a MAT SORT key, the "*" must be present in the same dimension of each sort key.

Error 340
Table Length Error

The length of the Special Case Table, stored in the 257th element of the LEXICAL ORDER array, must be in the range zero to sixty-three.

Error 341
Order Table Lower Byte Error

In a LEXICAL ORDER array, the lower byte of the first 256 entries indicates a special case. Legal values are explained in the *User's Guide*.

Error 342
Not a One-dimensional INTEGER Array

The array specified in the LEXICAL ORDER statement must be INTEGER and must have a RANK of one.

Error 343
Special Case Index is Too Big

The index points past the end of the special case table, whose length is specified in the 257th element of the array.

Error 344
2-to-1 List Length Error

In the special case table, a 2-to-1 list must start with a length. The length gives the number of entries in the list. You will get this error if the length is negative, zero or longer than the special case table.

Error 346
INDENT parameter out of range

The values specified in the INDENT statement are not legal.

Error 347
Structures improperly matched

The FOR...NEXT, LOOP...END LOOP, REPEAT...UNTIL, SELECT...END SELECT, WHILE...END WHILE, program structures are either nested improperly or there is a missing structured statement.

Error 401
Bad system function argument

A value passed to a system function was out of range or otherwise illegal. See the "Keyword Dictionary" chapter for this function for a description of legal values.

Error 427
Priority may not be lowered

When executing an error handling routine, the priority cannot be changed.

Error 435
EXEC not allowed on this Binary

The file is not an executable file or is corrupt.

Error 453
File in Use

The file or device is in use and this operation can not occur at this time.

Error 455
Possibly corrupt file

The executable file specified by EXECUTE is corrupt or is not an executable file, the file was locked, or the operating system is no longer recognizing the file as a valid, ASSIGNed file.

Error 456
Unsupported directory operation

The directory was specified in an illegal way, usually involving "." or "..".

Error 459
Specified file is not a directory

The specifier must refer to a directory, not a regular file. Or if the specifier includes a path, one of the directories specified in the path is not a directory.

Error 460
Directory not empty

The directory could not be deleted because files or sub-directories still exist in it.

Error 462
Invalid Password

An HP LIF style file password was started with the "<" character but no ">" character was found.

Error 465
Invalid rename across volumes

RENAME can not be used to move a file from one disk to another.

Error 471
TRANSFER not supported by Interface

TRANSFER is only supported on some devices. It is not supported on CRT, KBD, or parallel ports. If the device or interface is supposed to support TRANSFER, make sure the device driver is the current revision.

Error 481
File locked or open Exclusively

The file has already been ASSIGNed by yourself or another user and the file or part of the file is LOCKed for exclusive access. You may want to write a loop which tries the operation several times, waiting in between for the file to be UNLOCKed. Or you may want to LOCK the file yourself so that no one else can deny your access to it.

Error 482
Not allowed with a directory

Under DOS, a directory cannot be ASSIGNed.

Error 485
Invalid Volume Copy

The reasons for this error depend on your operating system. Copying a volume may not be supported on some systems.

Error 511
MAT INV result array must be REAL

The destination of a matrix invert operation must be a REAL array.

Error 543
Improper Dimensions for REDIM

The destination matrix could not be implicitly re-dimensioned by the MAT statement because the RANK of the destination matrix is not the same as the number of ranges specified in the array to the right of the equal sign.

Error 553

Cannot load object file

The object file being LOADed does not exist, has an invalid parameter, one of the library files needed to run this application could not be found, or the driver is not supported on the current Operating System.

Error 554

Object file not a widget

The file did not have a header which could be recognized as a widget.

Error 557

Undefined widget

The widget specified does not have internal (within HP BASIC for Windows) or external (WI prefixed file) code which the binary could locate.

Error 558

Undefined widget attribute

The widget attribute specified in conjunction with SET or RETURN is not valid.   See the list of valid attributes for the widget in question.

Error 559

Wrong parameter type for attribute

The parameter being passed to an attribute is of the wrong type. See the list of valid attributes for the widget in question.

Error 560

Menu not allowed in child widget

Menus are not allowed in a child widget.   Menus can only be children of a level-0 panel, or of another menu.

Error 561

Widget must have a parent

Widget cannot be created without a parent.

Error 562

Parent widget does not support this type of child

The parent widget does not allow this type of widget to be a child.   See if the widget can be used as a level-0 widget or as a child of another widget.

Error 563

SET not allowed for attribute

SET not allowed for attribute.

Error 564

RETURN not allowed for attribute

RETURN not allowed for attribute.

Error 565

VALUE out of range for attribute

Value out of range for attribute. Check the list of possible values for the attribute on the widget.

Error 566

Invalid value for attribute

Invalid value for attribute. The value may be in range, but this particular value is not allowed.

Error 567

Too few elements in array for attribute

Too few elements in array for attribute.   Make the array size larger.

Error 569

Invalid font specification

Invalid font specification.   See the FONT attribute under the widget being used.   Typical font specs look like “10 BY 20, BOLD”, etc.

Error 570

Undefined dialog type

A DIALOG type must be one of several pre-defined dialog box types. Typical types are “INFORMATION”, “WARNING”, AND “STRING”.

Error 571

Widget has no events to set

The widget has no events to set. Some widgets have no events associated with them. The widget causing this error is such a widget.

Error 572

Undefined widget event

The event specified is not one of the valid events for that particular widget. See the list of events for the widget in question.

Error 573

Attribute not available to child widget

Attribute not available to child widget.   See if the widget can be made a level-0 widget or if the attribute can be deleted.

Error 574

Attribute not available to level-0 widget

Attribute not available to level-0 widget.   See if the widget can be made a child of another widget or if the attribute can be deleted.

Error 602
Improper BUFFER Lifetime

It is an error to ASSIGN an I/O Path to a BUFFER if the BUFFER can cease to exist before the I/O Path. If the I/O Path is local, the BUFFER's lifetime will always equal or exceed the I/O Path's. If the I/O Path is in a COM block, the BUFFER must be in the same COM. If the I/O Path is a parameter, then the BUFFER must be in a COM block or must be a parameter also.

Error 603
Variable not declared BUFFER

The variable specified in the ASSIGN...TO BUFFER statement must be declared with the BUFFER keyword following it in the DIM, INTEGER, REAL or COM statement. If the buffer variable is a parameter, it must be passed with the BUFFER keyword following it in the DEF or SUB statement.

Error 604
Bad TRANSFER source or destination

Either the source or the destination, but not both, must be a BUFFER.

Error 606
Improper TRANSFER parameters

One of the following problems exists in the TRANSFER statement: DELIM can not be used on outbound transfers or if the I/O path has the WORD attribute. Or EOT was set to RECORD but no EOR was given to define a record.

Error 609
IVAL/DVAL result too large

The value in the string represents a number which is too large for the function to convert.

Error 611
Premature TRANSFER termination

An error occurred which caused the transfer to terminate abnormally.

Error 612
BUFFER pointers in use

The buffer pointer or count couldn't be changed because of an active transfer.

Error 620
Complex value not allowed here

This function does not handle complex values.

Error 623
ATN is undefined at +/- i

The ATN function is undefined at CMPLX(0,1) and CMPLX(0,-1).

Error 624
ACSH/ATNH argument out of range

The value specified is not within the legal range for the ACSH or ATNH functions.

Error 625
Bad SEARCH condition on Complex

This search condition is not allowed for complex arrays.

Error 700
Improper Plotter specifier

This plotter specifier is not supported or this interface is not legal for graphics output.

Error 704
Upper bound not greater than lower bound

The value of the upper clipping bound specified is lower than the value of the lower clipping bound.

Error 705
VIEWPORT/CLIP Beyond Hard Clip Limits

A value specified in the CLIP or VIEWPORT statement is too large or too small for the current graphic device.

Error 708
Device not initialized

The device is not the current PLOTTER IS or other active graphic device.

Error 713
Request not supported by device

This device does not support the requested operation.

Error 730
Internal error occurred in library call

A library or system call returned an unexpected error.

Error 733
GESCAPE opcode not recognized

The opcode specified is not supported on this device.

Error 810
Feature not supported on this system

This feature is not included in this release of HTBasic.

Error 815
Cannot access system time

The call to read the system time failed unexpectedly.

Error 826
EXECUTE process status failure

The process no longer exists and can not be killed.

Error 827
String too long for EXECUTE

Shorten the string and try again.

Error 831
Write to a broken pipe

OUTPUT on this I/O path is no longer allowed because the pipe to the process has been broken. The process probably terminated.

Error 832
Cannot seek on a pipe

The use of a record number with this I/O path is not allowed because the path refers to a pipe.

Error 833
Wrong direction data transfer in pipe

You can not ENTER from a pipe unless the pipe-specifier ends with the pipe character, "|". You can not OUTPUT to a pipe unless the pipe-specifier starts with the pipe character.

Error 841
CSUB run-time error

The CSUB called at this line encountered an error. Contact the supplier of the CSUB for more information.

Error 863
Not in a window system

This statement is not supported unless HTBasic is executing under a windowing system.

Error 898
Softkey Macro is too long

The length of the string must be less than 256 characters and there must be enough available macro memory to store it. LIST KEY reports the current amount of available softkey macro memory.

Error 899
Key number out of range

The specified key number is outside the legal range. See the CONFIGURE KEY statement.

Error 900
Undefined softkey macro

The key which you pressed does not presently have a softkey macro definition.

Error 901
Softkey Macro memory overflow

The available memory reserved for user defined Softkey Macro definitions is full.

Error 902
Must delete entire context

To delete a subprogram context or the SUB or FN statement of a subprogram context, all program lines in the SUB of DEF   context must be deleted.

Error 903
No line number room to renumber

A renumber operation would create line numbers larger than 4,194,304. (Note: The HP BASIC limit was 32766.)

Error 905
CHANGEd line too long

The CHANGE operation could not be completed because it would have created a line which is longer than 255 characters.

Error 906
SUB or DEF FN not allowed here

A new SUB or DEF FN must be created with a line number greater than all existing program lines.

Error 909
May not replace SUB or DEF FN

The SUB or DEF FN line delimits a context and so the SUB or DEF FN keywords can not be changed. Create a new context at the end of the program if necessary and use MOVELINES to move program lines to another context.

Error 910
Identifier not found in context

The specified identifier was not found in the current context. This error can also occur if an attempt is made to access a main context variable after adding a program line. Adding a program line causes the values of all variables to be discarded.

Error 935
Identifier too long

An identifier may be up to 15 characters in length.

Error 936
Unrecognized Character

A character in the program line was not legal. You probably mistyped an option in the LOAD BIN statement or that particular BIN doesn't support the option specified.

Error 937
Invalid OPTION BASE

The value specified was not zero or one.

Error 940
Duplicate formal parameter name

The parameter appears more than once in the formal parameter list.

Error 949
Syntax error at cursor

The item pointed to by the cursor is not valid in this position for this statement. See the "Keyword Dictionary" chapter entry for the correct syntax.

Error 951
Incomplete Statement or Command

There are more required items for this statement. See the "Keyword Dictionary" chapter entry for the correct syntax.

Error 956
Source/destination mismatch

The number of array elements do not match in the source and destination arrays.

Error 962
Programmable only

This statement may not be executed from the keyboard. It may only be stored and executed in a program.

Error 963
Command only

This statement may be executed from the keyboard only. It may not be stored or executed in a program.

Error 977
Statement or Command too complex

An expression in the statement is too complex. Either simplify the expression or split it into two or more expressions.

Error 980
Too many symbols in context

There are too many variables, I/O Paths and labels in the program context. Break the program into two or more SUBs or DEF FNs.

Error 985
Invalid Quoted String

The closing quote character is missing.

Error 987
Invalid Line Number

The program line number is outside the range of 1 through 4,194,304. (The HP BASIC limit was 32768.)

Error 1100

Unable to load DLL

The attempt to load the DLL specified failed. Either the DLL name given was incorrect (spelling) or The DLL is not located in the correct directory.

Error 1101

Unable to unload DLL

The system was unable to unload the DLL.

Error 1102

DLL is already loaded

A DLL by the specified name is already loaded. The DLL may have been loaded in a previous program or some error aborted the program before the UNLOAD command executed. It is not possible to load two DLL's of the same name even if they have different functions.

Error 1103

Unable to load Function

Basic was unable to load the specified function from the specified DLL. Verify that the DLL name and function names are correct. If you’re writing your own DLL, make sure function is exported.

Error 1104

Function is already loaded

A function by that name for the specified DLL is already loaded.

Error 1105

Alias previously used

The alias specified is already being used by another loaded function.   Aliases must be unique across all loaded DLL functions.

Error 1106

Invalid or no return type specified

A valid return type must be specified when doing the DLL Get.   See the DLL Loader document for a list of valid return types.

Error 1107

Invalid or no DLL name specified

When doing a DLL GET, the correctly loaded DLL must be specified.

Error 1108

Insufficient Dimension for passing string by reference

When passing a string by reference it must be dimensioned at least one byte larger than it will ever be.   If the string is using all of it's dimensioned space you get this error.

Error 1109

Unsupported Number of Parameters

The amount of data set up as parameters exceeds the 80-byte limit.

Error 1110

Possible HTBasic Memory Corruption

When a string is passed out by reference it is possible to write a string into HTBasic memory that exceeds the maximum dimension for the HTBasic variable. When HTBasic regains control from a Call to a DLL function that passes strings by reference, it attempts to detect occurrences of this and returns this error if any are found.

Error 2000
Stack Overflow

The processor stack has grown beyond the available memory. This is usually caused by user defined functions that are nested too deep.

Error 2001
Too many Open Files

HTBasic used to return 2001 for this condition. Now that HP BASIC has added error 189 for this condition, HTBasic has been changed to return 189 for compatibility.

Error 2002
HELP file not found

The HTB.HLP file was not found in the directory specified by the environment variable "HTB=xxx", in the current directory or in the same directory as HTB.EXE.

Error 2003
Bad Device Driver number

The CONFIGURE DEVICE statement, which returned this error, is no longer necessary and thus this error is not currently returned by HTBasic.

Error 2004
Bad Key Function number

The key function number specified is outside the legal range. See the CONFIGURE KEY statement.

Error 2005
Illegal in Run-only Version

This error is not currently returned by HTBasic.

Error 2006
Illegal DUMP device

This error is not currently returned by HTBasic. Error 56, "File Not Found," is returned when a CONFIGURE DUMP specifies a language for which no device driver file exists.

Error 2007
Wrong Object Type

An attempt was made to execute object code which is not suitable for the computer's processor. An HTBasic DOS 386/486 Version CSUB or BIN can not execute with the HTBasic DOS PC Version, etc.

Error 2008
May not modify CSUB

An attempt was made to change a CSUB definition.

Error 2009
Wrong Revision

The PROG or BIN file you attempted to LOAD, LOADSUB or CAT was created with an earlier release of HTBasic and is not compatible with the current release. For 1.x/2.x PROG files, use the HT2SAVE utility (explained in the *User's Guide*) to convert your PROG files to the current format. This can also be done by LOADing and SAVEing the file with the old release of HTBasic and then GETting and STOREing the file with the new release. For old BIN files, you must contact the supplier of the BIN file for information about upgrading.

Error 2010
May Not load driver Here

You must load all drivers from the MAIN program or as an immediate command when HTBasic is in the Idle condition. It is recommended that PLOTTER IS, CONFIGURE DUMP and GRAPHICS INPUT IS statements to load drivers be duplicated in the AUTOST file to insure the proper drivers are loaded before your programs begin to execute. LOAD BIN statements should also be executed in the AUTOST file.

Error 2011
Exceeded Graphics Driver Limit

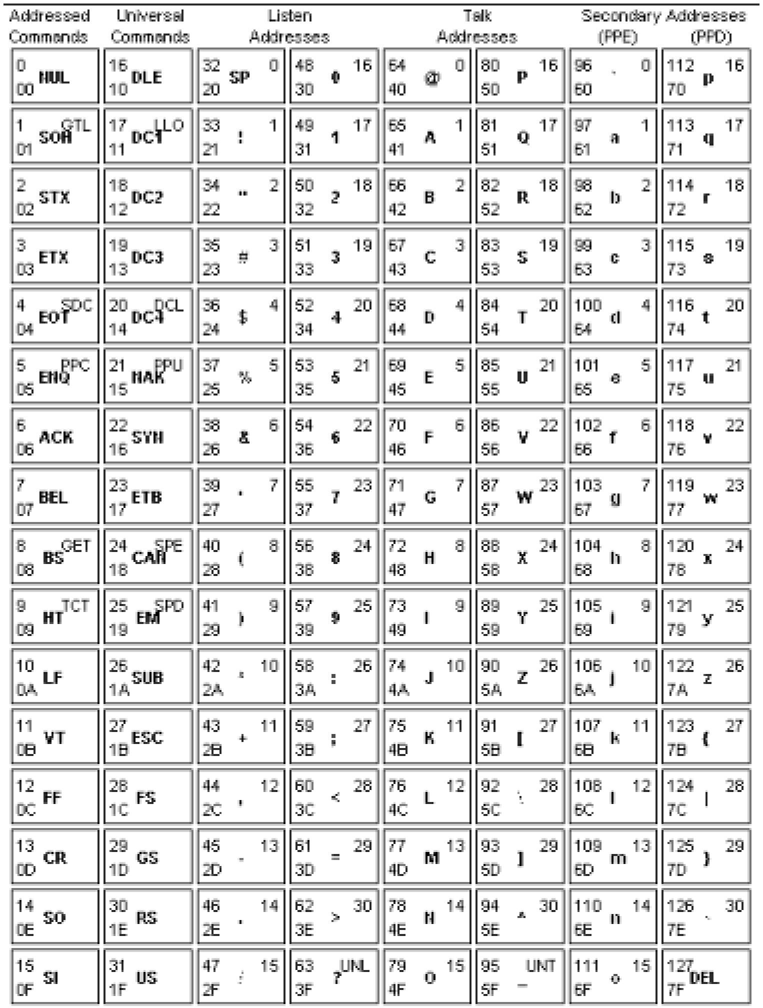
There is a limit to the number of device drivers which can be loaded with the CONFIGURE DUMP, GRAPHICS INPUT IS and PLOTTER IS statements. You have exceeded that limit, which is 10. Use LIST BIN to see a list of the currently loaded drivers.

Error 2012
Illegal CALL in CSUB

The CSUB attempted to CALL an interpreted SUB, which is not supported. Use "XREF sub-name : SB" to list the SUBs called by sub-name. Then make sure they are compiled or that no interpreted SUBs of the same name exist before the compiled SUBs.

Appendix B

ASCII Code Chart



Addressed Commands	Universal Commands	Listen Addresses		Talk Addresses		Secondary Addresses (PPE)	(PPD)
0 00 NUL	16 10 DLE	32 20 SP 0	48 30 0 16	64 40 @ 0	80 50 P 16	96 60 ` 0	112 70 p 16
1 01 SOH GTL	17 11 DC1 LLO	33 21 ! 1	49 31 1 17	65 41 A 1	81 51 Q 17	97 61 a 1	113 71 q 17
2 02 STX	18 12 DC2	34 22 " 2	50 32 2 18	66 42 B 2	82 52 R 18	98 62 b 2	114 72 r 18
3 03 ETX	19 13 DC3	35 23 # 3	51 33 3 19	67 43 C 3	83 53 S 19	99 63 c 3	115 73 s 19
4 04 EOT SDC	20 14 DC4 DCL	36 24 $ 4	52 34 4 20	68 44 D 4	84 54 T 20	100 64 d 4	116 74 t 20
5 05 ENQ PPC	21 15 NAK PPU	37 25 % 5	53 35 5 21	69 45 E 5	85 55 U 21	101 65 e 5	117 75 u 21
6 06 ACK	22 16 SYN	38 26 & 6	54 36 6 22	70 46 F 6	86 56 V 22	102 66 f 6	118 76 v 22
7 07 BEL	23 17 ETB	39 27 ' 7	55 37 7 23	71 47 G 7	87 57 W 23	103 67 g 7	119 77 w 23
8 08 BS GET	24 18 CAN SPE	40 28 (8	56 38 8 24	72 48 H 8	88 58 X 24	104 68 h 8	120 78 x 24
9 09 HT TCT	25 19 EM SPD	41 29) 9	57 39 9 25	73 49 I 9	89 59 Y 25	105 69 i 9	121 79 y 25
10 0A LF	26 1A SUB	42 2A * 10	58 3A : 26	74 4A J 10	90 5A Z 26	106 6A j 10	122 7A z 26
11 0B VT	27 1B ESC	43 2B + 11	59 3B ; 27	75 4B K 11	91 5B [27	107 6B k 11	123 7B { 27
12 0C FF	28 1C FS	44 2C , 12	60 3C < 28	76 4C L 12	92 5C \ 28	108 6C l 12	124 7C | 28
13 0D CR	29 1D GS	45 2D - 13	61 3D = 29	77 4D M 13	93 5D] 29	109 6D m 13	125 7D } 29
14 0E SO	30 1E RS	46 2E . 14	62 3E > 30	78 4E N 14	94 5E ^ 30	110 6E n 14	126 7E ~ 30
15 0F SI	31 1F US	47 2F / 15	63 3F ? UNL	79 4F O 15	95 5F _ UNT	111 6F o 15	127 7F DEL

Legend:
Center - ASCII Glyph or Mnemonic
Upper-left - Decimal
Upper-right - IEEE-488 Command or Address
Lower-left – Hexadecimal

ERROR

Six manual entries exist for ERROR.

See:

CAUSE ERROR	Simulates a specified error.
CLEAR ERROR	Resets all error indicators.
ERROR RETURN	Returns program execution to the line following the most recent error.
ERROR SUBEXIT	Returns subprogram execution to the line following the most recent error.
OFF ERROR	Cancels event branches defined by ON ERROR.
ON ERROR	Defines an event branch for trappable errors.

KEY

ITwelve manual entries exist for KEY.

See:

CONFIGURE KEY	Assigns editor functions to keyboard keys.
EDIT KEY	Puts you into softkey EDIT mode.
KEY LABELS	Controls the display of the softkey labels.
KEY LABELS PEN	Sets the color for the softkey labels.
LIST KEY	Lists the softkey macro definitions.
LOAD KEY	Loads softkey macro definitions into memory.
OFF KEY	Cancels event branches defined by ON KEY.
ON KEY	Defines an event branch for when a softkey is pressed.
SET KEY	Defines one or more softkey macros.
READ KEY	Returns one or more softkey macro definitions.
RE-STORE KEY	Stores the KEY definitions in a file.
STORE KEY	Stores the softkey definitions in a file.

CONFIGURE

Twelve manual entries exist for CONFIGURE.

See:

CONFIGURE BDAT	Specifies the byte order for CREATE BDAT.
CONFIGURE CREATE	Specifies the kind of file header used with typed files.
CONFIGURE DIM	Turns implicit variable dimensioning on or off.
CONFIGURE DUMP	Specifies what graphic printer language to use for DUMP.
CONFIGURE KBD	Defines keyboard mappings for character sets.
CONFIGURE KEY	Assigns editor functions to keyboard keys.
CONFIGURE LABEL	Defines characters for the LABEL statement.
CONFIGURE LONGFILENAMES	Specifies use of long filenames.
CONFIGURE MSI	Specifies HP style volume specifier translations.
CONFIGURE PRT	Specifies the value of PRT.
CONFIGURE SAVE	Sets the file type produced by SAVE.
CONFIGURE SYSTEM	Returns the conjugate of a complex number

REAL

Two manual entries exist for REAL.

See:

REAL Reserve floating point variable and and array storage.

REAL Converts an INTEGER or COMPLEX number to REAL.

SET

Nine manual entries exist for SET.

See:

SET ALPHA MASK	Determines which plane(s) can be modified by ALPHA display operations.
SET CHR	Defines the bit-patterns for one or more characters.
SET DISPLAY MASK	Specifies which planes can be seen on the alpha display.
SET ECHO	Sets the echo location on the PLOTTER IS device.
SET KEY	Defines one or more softkey macros.
SET LOCATOR	Sets a new graphic locator position on the GRAPHICS INPUT IS device.
SET PEN	Defines part or all of the color map.
SET TIME	Sets the time of day clock.
SET TIMEDATE	Sets the date and time of the computer's clock.

OFF event

Manual entries document each event separately.

See:

OFF CYCLE	Cancels a repeating event branch.
OFF DELAY	Cancels a single event branch after a specified number of seconds.
OFF END	Cancels an event branch for end-of-file conditions.
OFF EOR	Cancels an event branch for end-of-record conditions.
OFF EOT	Cancels an event branch for end-of-transfer conditions.
OFF ERROR	Cancels an event branch for trappable errors.
OFF EVENT	Cancels event branches defined by ON EVENT.
OFF INTR	Cancels a hardware interrupt initiated branch.
OFF KBD	Cancels an event branch for when a key is pressed.
OFF KEY	Cancels an event branch for when a softkey is pressed.
OFF KNOB	Cancels an event branch for when the KNOB is turned.
OFF SIGNAL	Cancels an event branch for when a SIGNAL statement is executed.
OFF TIME	Cancels a single event branch for a specific time.
OFF TIMEOUT	Cancels an event branch for an I/O timeout.

ON event

Manual entries document each event separately.

See:

ON	Transfers control to one of a list of lines.
ON CYCLE	Defines a repeating event branch.
ON DELAY	Defines a single event branch after a specified number of seconds.
ON END	Defines an event branch for end-of-file conditions.
ON EOR	Defines an event branch for end-of-record conditions.
ON EOT	Defines an event branch for end-of-transfer conditions.
ON ERROR	Defines an event branch for trappable errors.
ON EVENT	Defines an event branch that is taken after a widget generates that event.
ON INTR	Defines a hardware interrupt initiated branch.
ON KBD	Defines an event branch for when a key is pressed.
ON KEY	Defines an event branch for when a softkey is pressed.
ON KNOB	Defines an event branch for when the KNOB is turned.
ON SIGNAL	Defines an event branch for when a SIGNAL statement is executed.
ON TIME	Defines a single event branch for a specific time.
ON TIMEOUT	Defines an event branch for an I/O timeout.

ON EVENT

Defines an event branch that is taken after a widget generates that event.

Syntax: ON EVENT

where: action = { GOTO|GOSUB|RECOVER } line | CALL subprogram
line = line-number | line-label

Sample:

```
ON EVENT @Pushbutton_3,"ACTIVATED" GOSUB Clear_profile
ON EVENT @Slider, “DONE” GOSUB Change_temphigh
ON EVENT @Sldr,"CHANGED", VAL(SYSTEM$(“SYSTEM PRIORITY”))
+1 GOSUB Chg_stpnt
```

Description:

The ON EVENT statement not only sets up the ON EVENT branch, but also enables the event.

The most recent ON EVENT (or OFF EVENT) statement for a given widget and event combination overrides any previous ON EVENT definition for that combination. If the overriding ON EVENT definition occurs in a context different from the one in which the overridden ON EVENT occurs, the overridden ON EVENT is restored when the calling context is restored.

Any specified *line reference* for GOTO or GOSUB must be in the same context as the ON EVENT statement. CALL and GOSUB will return to the next line that would have been executed if the ON EVENT widget event had not been serviced. The system priority is restored to that which existed before the ON EVENT branch was taken.

RECOVER forces the program to go directly to the specified line in the context containing that ON EVENT statement. When RECOVER forces a change of context, the system priority is restored to that which existed in the original (defining) context at the time that context was exited.

NOTE

The priority specified in the ON EVENT statement (as in all ON-event statements) must be higher than the current system priority in order for the event to be recognized.

When you nest ON EVENT statements, be aware that the system priority is raised to the one you specified in the ON EVENT statement, when that event is serviced for CALL and GOSUB.

To ensure that the events are recognized for all of your ON EVENT statements, specify a higher priority each time you go deeper into the nesting. To do this, query for the current system priority and then increase it by one, instead of specifying the priority as a number between the event name and GOTO, GOSUB, RECOVER, or CALL.

Use the following command sequence within the ON EVENT statement
to do this. (This technique will cause an error if the current system priority is 15.)

```
VAL(SYSTEM$(“SYSTEM PRIORITY”))+1
```

CALL and RECOVER remain active when the context changes to a subprogram or function, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON EVENT is disabled by DISABLE EVENT or DISABLE, is re-enabled
by ENABLE EVENT or ENABLE, and is deactivated by OFF EVENT.

See Also:

DISABLE, DISABLE EVENT, ENABLE, ENABLE EVENT, OFF EVENT

Chapter 3

Statement Summary

The following table lists all the HTBasic keywords and indicates which statements can be executed from the keyboard, stored in a program, and included in an IF...THEN statement.

Letter	**Meaning**
K	Keyboard executable
P	Programmable
I	Legal in an IF...THEN

ABORT	KPI
ABORTIO	KPI
ABS	KPI
ACS	KPI
ACSH	KPI
ALLOCATE	KPI
ALPHA	KPI
ALPHA HEIGHT	KPI
ALPHA PEN	KPI
AND	KPI
AREA	KPI
ARG	KPI
ASN	KPI
ASNH	KPI
ASSIGN	KPI
ATN	KPI
ATN2	KPI
ATNH	KPI
AXES	KPI
BASE	KPI
BEEP	KPI
BINAND	KPI
BINCMP	KPI
BINEOR	KPI
BINEQV	KPI
BINIMP	KPI
BINIOR	KPI
BIT	KPI
BREAK	KPI
CALL	KPI
CASE	-P-
CAT	KPI
CAUSE	KPI
CHANGE	K--
CHECKREAD	KPI
CHGRP	KPI
CHOWN	KPI
CHR$	KPI
CHRX	KPI
CHRY	KPI

CINT	KPI
CLEAR	KPI
CLEAR ERROR	-PI
CLEAR LINE	KPI
CLEAR SCREEN	KPI
CLIP	KPI
CLS	KPI
CMPLX	KPI
COM	-P-
COMMAND$	KPI
COMPLEX	-P-
CONFIGURE	KPI
CONJG	KPI
CONT	K--
CONTROL	KPI
COPY	KPI
COPYLINES	K--
COS	KPI
COSH	KPI
CREATE	KPI
CREATE ASCII	KPI
CREATE BDAT	KPI
CREATE DIR	KPI
CRT	KPI
CSIZE	KPI
CSUB	---
DATA	-P-
DATE	KPI
DATE$	KPI
DEALLOCATE	KPI
DEF FN	-P-
DEG	KPI
DEL	K--
DELSUB	KPI
DET	KPI
DIALOG	KPI
DIGITIZE	KPI
DIM	-P-
DISABLE	KPI
DISABLE EVENT	KPI
DISABLE INTR	KPI
DISP	KPI
DISPLAY FUNCTIONS	KPI
DIV	KPI
DLL GET	KPI
DLL LOAD	KPI
DLL READ	KPI
DLL UNLOAD	KPI
DLL WRITE	KPI

DOT	KPI
DRAW	KPI
DROUND	KPI
DUMP	KPI
DUMP DEVICE IS	KPI
DVAL	KPI
DVAL$	KPI
EDIT	K--
EDIT KEY	K--
ELSE	-P-
ENABLE	KPI
ENABLE EVENT	KPI
ENABLE INTR	KPI
END	-P-
ENTER	KPI
ENVIRON$	KPI
ERRL	-PI
ERRLN	KPI
ERRM$	KPI
ERRN	KPI
ERROR	-PI
EXECUTE string	KPI
EXIT IF	-P-
EXOR	KPI
EXP	KPI
FIND	K--
FIX	KPI
FN	KPI
FNEND	-P-
FOR	-P-
FRACT	KPI
FRAME	KPI
FRE	KPI
GCLEAR	KPI
GESCAPE	KPI
GET	KPI
GFONT IS	KPI
GINIT	KPI
GLOAD	KPI
GOSUB	-PI
GOTO	-PI
GRAPHICS	KPI
GRAPHICS INPUT IS	KPI
GRID	KPI
GSEND	KPI
GSTORE	KPI
HELP	K--
IDRAW	KPI
IF	-P-

IMAG	KPI
IMAGE	-P-
IMOVE	KPI
INDENT	K--
INITIALIZE	KPI
INMEM	KPI
INP	KPI
INPUT	-PI
INPW	KPI
INT	KPI
INTEGER	-P-
IPLOT	KPI
IVAL	KPI
IVAL$	KPI
KBD	KPI
KBD CMODE	KPI
KBD LINE PEN	KPI
KBD$	KPI
KEY LABELS	KPI
KEY LABELS PEN	KPI
KNOBX	KPI
KNOBY	KPI
LABEL	KPI
LDIR	KPI
LEN	KPI
LET	KPI
LEXICAL ORDER IS	KPI
LGT	KPI
LINE TYPE	KPI
LINK	KPI
LINPUT	-PI
LIST	KPI
LIST BIN	KPI
LIST DLL	KPI
LIST KEY	KPI
LOAD	KPI
LOAD BIN	KPI
LOAD KEY	KPI
LOADSUB	KPI
LOCAL	KPI
LOCAL LOCKOUT	KPI
LOCK	KPI
LOG	KPI
LONG	-P-
LOOP	-P-
LORG	KPI
LWC$	KPI
MASS STORAGE IS	KPI
MAT	KPI

MAT REORDER	KPI
MAT SEARCH	KPI
MAT SORT	KPI
MAX	KPI
MAXLEN	KPI
MAXREAL	KPI
MERGE ALPHA	KPI
MIN	KPI
MINREAL	KPI
MOD	KPI
MODULO	KPI
MOVE	KPI
MOVELINES	K--
MSI	KPI
NEXT	-P-
NOT	KPI
NPAR	KPI
NUM	KPI
ON---GOTO/GOSUB	-PI
ON/OFF CDIAL	-PI
ON/OFF CYCLE	-PI
ON/OFF DELAY	-PI
ON/OFF END	-PI
ON/OFF EOR	-PI
ON/OFF EOT	-PI
ON/OFF ERROR	-PI
ON/OFF EVENT	-PI
ON/OFF INTR	-PI
ON/OFF KBD	-PI
ON/OFF KEY	-PI
ON/OFF KNOB	-PI
ON/OFF SIGNAL	-PI
ON/OFF TIME	-PI
ON/OFF TIMEOUT	-PI
OPTION BASE	-P-
OR	KPI
OUT	KPI
OUTPUT	KPI
OUTW	KPI
PASS CONTROL	KPI
PAUSE	KPI
PDIR	KPI
PEN	KPI
PENUP	KPI
PERMIT	KPI
PI	KPI
PIVOT	KPI
PLOT	KPI
PLOTTER IS	KPI

POLYGON	KPI
POLYLINE	KPI
POS	KPI
PPOLL	KPI
PRINT	KPI
PRINT LABEL	KPI
PRINT PEN	KPI
PRINTALL IS	KPI
PRINTER IS	KPI
PROTECT	KPI
PROUND	KPI
PRT	KPI
PURGE	KPI
QUIT	KPI
RAD	KPI
RANDOMIZE	KPI
RANK	KPI
RATIO	KPI
RE-SAVE	KPI
RE-STORE	KPI
RE-STORE KEY	KPI
READ	KPI
READ KEY	KPI
READ LABEL	KPI
READ LOCATOR	KPI
READIO	KPI
REAL	-P-
REAL()	KPI
RECTANGLE	KPI
REDIM	KPI
REM	-P-
REMOTE	KPI
REN	K--
RENAME	KPI
REPEAT	-P-
REQUEST	KPI
RES	KP-
RESET	KPI
RESTORE	-PI
RESUME	KPI
RETURN	-PI
REV$	KPI
RND	KPI
ROTATE	KPI
RPLOT	KPI
RPT$	KPI
RUN	K--
RUNLIGHT	KPI
SAVE	KPI

SC	KPI
SCRATCH A/ALL	K--
SCRATCH B/BIN	K--
SCRATCH C/COM	K--
SCRATCH KEY	K--
SCRATCH R/RECALL	K--
SECURE	K--
SELECT	-P-
SEND	KPI
SEPARATE ALPHA	KPI
SET ALPHA MASK	KPI
SET CHR	KPI
SET DISPLAY MASK	KPI
SET ECHO	KPI
SET KEY	KPI
SET LOCATOR	KPI
SET PEN	KPI
SET TIME	KPI
SET TIMEDATE	KPI
SGN	KPI
SHIFT	KPI
SHOW	KPI
SIGNAL	KPI
SIN	KPI
SINH	KPI
SIZE	KPI
SOUND	KPI
SPOLL	KPI
SQR	KPI
SQRT	KPI
STATIC	-P-
STATUS	KPI
STATUS()	KPI
STOP	KPI
STORE	KPI
STORE KEY	KPI
STORE SYSTEM	K--
SUB	-P-
SUBEND	-P-
SUBEXIT	-PI
SUM	KPI
SUSPEND	KPI
SYMBOL	KPI
SYSTEM KEYS	KPI
SYSTEM PRIORITY	KPI
SYSTEM$	KPI
TAN	KPI
TANH	KPI
TIME	KPI

TIME$	KPI
TIMEDATE	KPI
TIMEZONE IS	KPI
TRACE	KPI
TRACK	KPI
TRANSFER	KPI
TRIGGER	KPI
TRIM$	KPI
UNLOCK	KPI
UNTIL	-P-
UPC$	KPI
USER KEYS	KPI
VAL	KPI
VAL$	KPI
VIEWPORT	KPI
WAIT	KPI
WAIT FOR EOR	KPI
WAIT FOR EOT	KPI
WAIT FOR EVENT	KPI
WHERE	KPI
WHILE	-P-
WILDCARDS	KPI
WINDOW	KPI
WRITEIO	KPI
XREF	K--

Default FORMAT Chart

Target	ASSIGN (no FORMAT option)	ASSIGN; FORMAT ON	ASSIGN; FORMAT OFF	ASSIGN; FORMAT LSB FIRST	ASSIGN; FORMAT MSB FIRST
Ordinary file	Ordinary * Binary	Ordinary ASCII	Ordinary * Binary	Ordinary LSB Binary	Ordinary MSB Binary
ASCII File	LIF ASCII	LIF ASCII	LIF ASCII	LIF ASCII	LIF ASCII
BDAT File	BDAT † Binary	BDAT ASCII	BDAT † Binary	BDAT LSB Binary	BDAT MSB Binary
Device	ASCII	ASCII	MSB Binary	LSB Binary	MSB Binary
BUFFER	ASCII	ASCII	* Binary	LSB Binary	MSB Binary
String ‡					

† The byte order used with a BDAT file is established when the file is created and FORMAT OFF should be used to specify binary data. CONFIGURE BDAT is used to set the byte order for CREATE BDAT.

‡ Although you can't ASSIGN to a non-BUFFER string, you can OUTPUT/ENTER to any string. In these cases, the format is always ASCII.

* The native byte order for the computer is used. Using the native byte order for a computer results in faster throughput.

ABORT EXAMPLE

```
10    CLEAR SCREEN
20    RESET 7
30    PRINT STATUS(7,3)  ! 213 indicates active system controller at address
21
40    PASS CONTROL 720   ! pass control to another machine with notsys
50    WAIT .1
60    PRINT STATUS(7,3)  ! 149 indicates system controller but not active
controller
70    ABORT 7            ! should restore system controller status
80    PRINT STATUS(7,3)  ! 213 indicates active system controller at address
21
90    END
```

ABORTIO Example

```
10    DIM Buff$[2000] BUFFER
20    CLEAR SCREEN
30    PRINT "*** ABORTIO TEST ****"
40    PRINT
50    PRINT
60    PRINT "  Press F1 to cause an ABORTIO"
70    ASSIGN @Test TO BUFFER [50]
80    ASSIGN @In TO 9
90    TRANSFER @In TO @Test
100   I=0
110   LOOP
120     WAIT .1
130     ON KEY 1 GOTO 150
140     GOTO 160
150     ABORTIO @In
160     IF (STATUS(@Test,10))=64 THEN
170       DISP "Transfer status is: Active"
180     ELSE
190       DISP "Transfer status is: Inactive"
191       GOTO 230
200     END IF
210     I=I+1
220   END LOOP
230   END
```

ABS EXAMPLE

```
10    CLEAR SCREEN
20    X=-7
30    PRINT "The absolute value of ";X;"is:";ABS(X) !Takes the absolute value.
40    PRINT
50    COMPLEX C
60    A=4
70    B=3
80    C=CMPLX(A,B)
90    PRINT "The magnitude of the complex number:";A;"+";B;"i"
100   PRINT "is:";ABS(C)  !Takes the magnitude of the complex number.
110   END
```

ACS EXAMPLE

```
10! Test triangle leg 1 = 3 units, leg 2 = 4 units, hyp. = 5
20! units. Find the
30! angle between leg 2 and hyp. /|
40!                           5 / |3
50!                            /  |
60!                           -----
70!                             4
80!   The ACS function returns the angle between, in this case
90! leg 2 and the hyp. Take the adjacent leg (4) over the
100! hyp. (5) -- which is the cosine. ACS is the opposite
110! of COS.
120   CLEAR SCREEN
130   PRINT "The angle, using arccosine function."
140   RAD
150   PRINT "is";ACS(4/5);"radians."
160   DEG
170   PRINT "or";ACS(4/5);"degrees."
180   END
```

ACHS EXAMPLE

```
10    COMPLEX C
20    C=CMPLX(4,7)
30    CLEAR SCREEN
40    X=ACSH(C)
50    Y=LOG(C+CMPLX(0,1)*SQR(1-C^2))
60    PRINT X;"=";Y;"?"
70    IF X=Y THEN
80      PRINT "True"
90    ELSE
100     PRINT "False"
110   END IF
120   END
```

ALLOCATE EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Total Memory "&SYSTEM$("AVAILABLE MEMORY")
30    PRINT "***ALLOCATE TEST #4***"
40    ALLOCATE A$[32000],B(1000),INTEGER C(10)
50    PRINT "Memory after allocation "&SYSTEM$("AVAILABLE MEMORY")
60    DEALLOCATE A$,C(*),B(*)
70    PRINT "Memory freed "&SYSTEM$("AVAILABLE MEMORY")
80    ALLOCATE A$[32000],B(1000)
90    PRINT "Memory allocated without the integer array "&SYSTEM$("AVAILABLE MEMORY")
100   PRINT
110   PRINT "Memory before SUB call "&SYSTEM$("AVAILABLE MEMORY")
120   PRINT "Calling SUB..."
130   Yahoo
140   PRINT "Memory after SUB call "&SYSTEM$("AVAILABLE MEMORY")
150   PRINT
160   ALLOCATE Str$[90]
170   PRINT "Memory after allocated string "&SYSTEM$("AVAILABLE MEMORY")
180   ALLOCATE REAL D(8)
190   PRINT "Memory after allocated REAL array "&SYSTEM$("AVAILABLE MEMORY")
200   END
210     !
220   SUB Yahoo
230     PRINT "Memory in SUB call "&SYSTEM$("AVAILABLE MEMORY")
240     ALLOCATE COMPLEX Y(4)
250     PRINT "Memory after allocation of COMPLEX array "&SYSTEM$("AVAILABLE MEMORY")
260   SUBEND
```

ALPHA HEIGHT EXAMPLE

```
10    CLEAR SCREEN
20    ALPHA HEIGHT 10! error in documentation
30    FOR Loop=1 TO 30
40      PRINT Loop
50    NEXT Loop
60    PRINT "Paused...",STATUS(CRT,3) !Get the # of lines in the extended
output area.
70    PAUSE
80    DISP "Test complete"
90    ALPHA HEIGHT ! return to normal
100   END
```

ALPHA PEN EXAMPLE

```
10    PLOTTER IS CRT,"INTERNAL";COLOR MAP !Turn on color map mode.
20    CLEAR SCREEN
30    FOR Loop=0 TO 15
40      ALPHA PEN Loop     !This prints out the ALPHA PEN # in its proper
color.
50      PRINT Loop
60    NEXT Loop
70    PRINT "Test complete"
80    END
```

AND EXAMPLE

```
10    DATA 0,0,0,1,1,0,1,1
20    RESTORE
30    CLEAR SCREEN
40    PRINT "AND test"
50    PRINT " J"," K","J AND K"
60    FOR L=1 TO 4
70      READ J,K           !Reads in the DATA values into J and K.
80      PRINT J,K,J AND K  !Performs the AND operation, then prints it out.
90    NEXT L
100   END
```

AREA COLOR EXAMPLE

```
10    GINIT
20    GCLEAR
30    PLOTTER IS CRT,"INTERNAL";COLOR MAP
40    WINDOW 0,1.31,1.31,0
50    FOR L=0 TO 1 STEP .2
60      FOR H=0 TO 1 STEP 1/6
70        FOR S=0 TO 1 STEP .2
80          AREA COLOR H,S,L
90          MOVE H+.11*L,S+.11*L
100         RECTANGLE .09,.1,FILL,EDGE
110       NEXT S
120     NEXT H
130   NEXT L
140   LORG 7
150   MOVE 1.2,1.31
160   LABEL "z = Luminosity"
170   LORG 4
180   MOVE .6,1.31
190   LABEL "x = Hue"
200   LDIR PI/2
210   MOVE 1.31,.6
220   LABEL "y = Saturation"
230   END
```

AREA PEN EXAMPLE

```
10    GINIT
20    GCLEAR
30    PLOTTER IS CRT,"INTERNAL";COLOR MAP  !Set it to color map mode.
40    MOVE 40,40
50    FOR L=1 TO 40
60      AREA PEN L        !Show the different AREA PEN colors.
70      RECTANGLE 90,30,FILL,EDGE
80      LABEL L;
90      MOVE 40,40
100     WAIT 1
110     GCLEAR
120   NEXT L
130   END
```

ARG EXAMPLE

```
10    ! The ARG function returns the ANGLE of a complex number.
20    ! This program computes the angle and compares it with the result of ARG.
30    DEG
40    CLEAR SCREEN
50    COMPLEX A
60    A=CMPLX(3,4)
70    Mag=ABS(A)
80    Angle=ARG(A)
90    X=Mag*COS(Angle)
100   Y=Mag*SIN(Angle)
110   PRINT "Complex number a + bi: ";A
120   PRINT "Magnitude: ";Mag,"Angle: ";Angle
130   PRINT "Your complex number was: ";X;"+";Y;"i"
140   END
```

ASN EXAMPLE

```
10! Test triangle leg 1 = 3 units, leg 2 = 4 units, hyp. = 5
20! units. Find the
30! angle between leg 2 and hyp. /|
40!                           5 / |3
50!                            /  |
60!                           -----
70!                             4
80!   The ASN function returns the angle between, in this case
90! leg 2 and the hyp. Take the opposite leg (3) over the
100! hyp. (5) -- which is the sine. ASN is the opposite
110! of SIN.
120   CLEAR SCREEN
130   PRINT "The angle, using arcsine function."
140   RAD
150   PRINT "is";ASN(3/5);"radians."
160   DEG
170   PRINT "or";ASN(3/5);"degrees."
180   END
```

ASNH EXAMPLE

```
10    COMPLEX C
20    C=CMPLX(4,7)
30    CLEAR SCREEN
40    X=ASNH(C)             !Takes the hyperbolic arcsine of complex number C.
50    Y=LOG(C+SQR(C^2+1))   !Uses the mathematical approach to the hyperbolic
arcsine.
60    PRINT X;"=";Y;"?"
70    IF X=Y THEN           !Compare X and Y to verify ASNH.
80      PRINT "True"
90      STOP
100   ELSE
110     PRINT "False"
120   END IF
130   END
```

ASSIGN EXAMPLE

```
10    ON ERROR GOTO 50
20    CLEAR SCREEN
30    PRINT "*** output TEST ****"
40    PURGE "test.txt"
50    CREATE "test.txt",0
60    ASSIGN @File TO "test.txt";FORMAT ON
70    A$="This is a test."
80    OUTPUT @File;A$
90    RESET @File
100   ENTER @File;Test$
110   ASSIGN @File TO *
120   IF A$=Test$ THEN
130     PRINT "Test passed."
140     PURGE "test.txt"
150   ELSE
160     PRINT "Test failed."
170     PRINT "Output string did not equal input string."
180   END IF
190   END
```

ATN EXAMPLE

```
10! Test triangle leg 1 = 3 units, leg 2 = 4 units, hyp. = 5
20! units. Find the
30! angle between leg 2 and hyp. /|
40!                           5 / |3
50!                            /  |
60!                           -----
70!                             4
80!   The ATN function returns the angle between, in this case
90! leg 2 and the hyp. Take the opposite leg (3) over the
100! adjasent (4) -- which is the tangent. ATN is the opposite
110! of TAN.
120   CLEAR SCREEN
130   PRINT "The angle, using arctangent function."
140   RAD
150   PRINT "is";ATN(3/4);"radians."
160   DEG
170   PRINT "or";ATN(3/4);"degrees."
180   END
```

ATNH EXAMPLE

```
10    COMPLEX C
20    C=CMPLX(4,7)
30    CLEAR SCREEN
40    X=ATNH(C)              !Returns the hyperbolic arctangent of a complex
number.
50    Y=1/2*LOG((1+C)/(1-C)) !Performs the mathematical value for hyperbolic
arctangent.
60    PRINT X;"=";Y;"?"
70    IF X=Y THEN            !Verifies ATNH function.
80      PRINT "True"
90    ELSE
100     PRINT "False"
110   END IF
120   END
```

ATN2 EXAMPLE

```
10    !ATN2 returns the angle of the vector from the origin to a point.
20    !In this case, -4 is the y-value and 3 is the x-value.
30    CLEAR SCREEN
40    PRINT ATN2(-4,3)
50    END
```

AXES EXAMPLE

```
10    KEY LABELS OFF
20    CLEAR SCREEN
30    AXES
40    PRINT "Default Axes"
50    Holdit
60    AXES 5
70    PRINT "Ticks on X axes"
80    Holdit
90    AXES 5,5              !X,Y tick spacing.
100  PRINT "with 5,5 tick spacing"
110  Holdit
120  AXES 5,5,20
130  PRINT "X origin at 20"
140  Holdit
150  AXES 5,5,20,20         !20,20 is the X,Y origin of axis on the screen.
160  PRINT "X and Y Origins at 20,20"
170  Holdit
180  AXES 2,5,20,20,5
19   PRINT "X tick spacking 2 Y tick spacing 5, Origin at 20,20 major tick
size 5"
200  Holdit
210  AXES 2,2,20,20,5,5     !5,5 is the spacing of major ticks on X and Y
axis.
220  PRINT "5,5 major tick spacing"
230  Holdit
240  AXES 2,2,20,20,5,5,2   !2 is the size of major ticks.
250  PRINT "Major tick size of 2"
260  KEY LABELS ON
270  END
280  SUB Holdit
290    PRINT TABXY(50,50);"Press Continue"
300    PAUSE
310    CLEAR SCREEN
320    DISP
330  SUBEND
```

BASE EXAMPLE

```
10    OPTION BASE 1
20    DIM A(16,6)
30    CLEAR SCREEN
40    Pass_a(A(*))
50    PRINT "After REDIM"
60    REDIM A(7,3)
70    Pass_a(A(*))
80    END
90    SUB Pass_a(REAL A(*))
100    L=RANK(A)
110    PRINT "The array passed in, has the following dimensions."
120    PRINT "Rank: ";L
130    PRINT "Base: ";BASE(A,L)
140    PRINT "Size: ";SIZE(A,L)
15   SUBEND
```

BDAT EXAMPLE

```
10    CLEAR SCREEN
20    CREATE BDAT "stuff1.BDT",67    !Create a BDAT file.
30    CREATE BDAT "stuff2.BDT",67,78 !Create a BDAT file.
40    CAT "*.BDT"                    !Make sure the files were created.
50    PRINT "Press CONTINUE to purge files..."
60    PAUSE
70    PURGE "stuff1.BDT"
80    PURGE "stuff2.BDT"
81    CAT "*.BDT"
90    END
```

BEEP EXAMPLE

```
10    CLEAR SCREEN
20    BEEP
30    PRINT "*** BEEP test ***"
40    Duration=.01
50    FOR Frequency=81 TO 5208 STEP 10
60      BEEP Frequency, Duration         !Test BEEP at different frequencies.
70    NEXT Frequency
80    PRINT "Done."
90    END
```

BIN EXAMPLE

```
10    CLEAR SCREEN
20    LIST BIN     !List the binaries loaded in your computer.
30    END
```

BINAND EXAMPLE

```
10    INTEGER X,Y,Z
20    X=12
30    Y=6
40    Z=BINAND(X,Y)  !Do a binary AND of X and Y.
50    PRINT X;" & ";Y
60    See(Z)
70    END
80    SUB See(INTEGER X)
90      FOR Loop=15 TO 0 STEP -1
100       Temp=BIT(X,Loop)  !Print out the answer in bits.
110       PRINT Temp;
120     NEXT Loop
130   SUBEND
```

BINCMP EXAMPLE

```
10    INTEGER X,Y
20    X=13
30    Y=BINCMP(X)  !Binary Complement of X.
40    PRINT "Compliment of X"
50    See(Y)
60    END
70    SUB See(INTEGER X)
80      FOR Loop=15 TO 0 STEP -1
90        Temp=BIT(X,Loop)
100       PRINT Temp;
110     NEXT Loop
120   SUBEND
```

BINEOR EXAMPLE

```
10    INTEGER X,Y
20    X=12
30    Y=BINEOR(X,6)  !Binary EXOR of X and 6.
40    PRINT "X EXOR 6"
50    See(Y)
60    END
70    SUB See(INTEGER X)
80      FOR Loop=15 TO 0 STEP -1
90        Temp=BIT(X,Loop)
100       PRINT Temp;
110     NEXT Loop
120   SUBEND
```

BINEQV EXAMPLE

```
10    INTEGER X,Y
20    X=12
30    Y=BINEQV(X,6)  !Performs a bit by bit equivalence operation.
40    PRINT "Equivalence test"
50    See(Y)
60    END
70    SUB See(INTEGER X)
80      FOR Loop=15 TO 0 STEP -1
90        Temp=BIT(X,Loop)
100       PRINT Temp;
110     NEXT Loop
120   SUBEND
```

BINIMP EXAMPLE

```
10    INTEGER X,Y
20    X=12
30    Y=BINIMP(X,6)   !Performs a bit by bit implication operation.
40    PRINT "Implication test"
50    See(Y)
60    END
70    SUB See(INTEGER X)
80      FOR Loop=15 TO 0 STEP -1
90        Temp=BIT(X,Loop)
100       PRINT Temp;
110     NEXT Loop
120   SUBEND
```

BINIOR EXAMPLE

```
10    INTEGER X,Y
20    X=12
30    Y=BINIOR(X,6)  !Performs an OR operation of 12 and 6.
40    PRINT "12 OR 6"
50    See(Y)
60    END
70    SUB See(INTEGER X)
80      FOR Loop=15 TO 0 STEP -1
90        Temp=BIT(X,Loop)
100       PRINT Temp;
110     NEXT Loop
120   SUBEND
```

BIT EXAMPLE

```
10    INTEGER X
20    X=12
30    PRINT "The bits for ";X;":"
40    See(X)
50    END
60    SUB See(INTEGER X)
70      FOR Loop=15 TO 0 STEP -1
80        Temp=BIT(X,Loop)
90        PRINT Temp;
100     NEXT Loop
110   SUBEND
```

BREAK EXAMPLE

Assumes two com ports connected via null modem cable.

```
FOR COM2

10    ! LOAD BIN "SERIAL32;DR 2 DIS"
20    LOOP
30    PRINT STATUS (11,10) ! WILL BE 96 WHEN IDLE
40    IF BIT(STATUS (11,1),4) THEN GOTO 60
50    END LOOP
60    PRINT "BREAK" ! Break Detected
70    END

FOR COM1

10    ! LOAD BIN "SERIAL32; DR 1 DIS"
20    CONTROL 9,1;1 ! SENDS 400 MS BREAK
30    END
```

BUFFER EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** Assign Buffer TEST ****"
30    ASSIGN @Test TO BUFFER [2000]
40    A$="This is a test."
50    OUTPUT @Test;A$        !Output to the Buffer string.
60    ENTER @Test;Msg$       !Enter from the Buffer string.
70    IF A$=Msg$ THEN PRINT "Test passed."
80    PRINT "All done."
90    END
```

BYTE EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** output TEST ****"
30    CREATE "test.txt",0
40    ASSIGN @File TO "test.txt";BYTE   !Send the data out as 8-bit bytes.
50    A$="This is a test."
60    OUTPUT @File;A$
61    RESET @File
70    ENTER @File;Test$
80    IF A$=Test$ THEN PRINT "Test passed."
90    ASSIGN @File TO *
100   PURGE "test.txt"
110   END
```

CALL EXAMPLE

```
10    CLEAR SCREEN
20    CALL Msg("Hello world",1)  !These parameters will be passed down to the
SUB.
30    Msg("This is line two",2)
40    CALL "Msg" WITH ("Line three",3)
50    END
60    SUB Msg(Msg$,INTEGER X)
70      PRINT PEN X
80      PRINT Msg$
90    SUBEND
```

CASE EXAMPLE

```
10    CLEAR SCREEN
20    INPUT "Please enter your age:",Age
30    SELECT Age
40    CASE <1,>100
50      PRINT "Congratulations - Movie is free!"
60    CASE <12
70      Price=2.00
80    CASE 12 TO 59
90      Price=6.50
100   CASE 60
110     PRINT "Special movie rate"
120     Price=3.00
130   CASE ELSE
140     Price=4.50
150   END SELECT
160 Image: IMAGE "Movie price is $", D.2D
170   PRINT USING Image;Price
180 End: END
```

CAT EXAMPLE

```
10    DIM Files$(80)[80]
20    CONFIGURE LONGFILENAMES OFF
25    CONFIGURE LONGCATDATES OFF
30    Directory$="C:\"
40    CLEAR SCREEN
50    PRINT "Major CAT test program instructions: "
60    PRINT
70    PRINT
80    PRINT "Type CONT to continue"
90    PRINT "when paused."
100  DISP "*** Cat TEST ****"
110  New("Regular CAT")
120  CAT Directory$
130
140  New("CAT to printer")
150  CAT Directory$ TO #10  !CAT to the Windows default printer.
160  PRINTER IS CRT
170
180  CAT Directory$;NO HEADER   !CAT without the header.
190
200  New("Just the file names")
210  CAT Directory$;NAMES       !CAT the names of the files only.
220
230  New("Extend option")
240  CAT Directory$;EXTEND      !CAT with the EXTEND option to suppress the
SRM format.
250
260  New("Long file names")
270  CONFIGURE LONGFILENAMES ON
280  CAT Directory$             !CAT with longfilenames on.
290  CONFIGURE LONGFILENAMES OFF
300
310  New("CAT *.txt files.")
320  CAT Directory$&"*.txt"     !Look for only the files with *.txt
extensions.
330
340  New("CAT *.txt to string test")
350  CAT Directory$&"*.txt" TO Files$(*)   !CAT to a file.
360  PRINT "The first 10 text files in "&Directory$
370  FOR Loop=1 TO 10
380    PRINT Loop,File$(Loop)
390  NEXT Loop
```

```
400
410  New("Number of files")
420  CAT Directory$;NAMES,COUNT Count  !Find the number of files in the
directory.
430  PRINT "The number of files in "&Directory$&" is";Count
440
450  New("CAT the file names, skiping the first 10")
460  CAT Directory$;NAMES,SKIP 10   !CAT skipping the first ten files.
470
480  New("Files that start with 'T'")
490  CAT Directory$;SELECT "T"      !CAT only the files starting with T.
500  New("Test complete")
510  END
520  SUB New(Msg$)
530    PAUSE
540    CLEAR SCREEN
550    DISP Msg$
560  SUBEND
```

CAUSE ERROR EXAMPLE

```
10    CLEAR SCREEN
20    ON ERROR RECOVER Here
30    PRINT "Calling Force error SUB."
40    Force_error
50    PRINT "If it reaches here, it didn't work."
60 Here: PRINT "It works!"
70    END
80    !
90    SUB Force_error
100     PRINT "Forcing error 1"
110     CAUSE ERROR 1
120   SUBEND
```

CD EXAMPLE

```
10  ! Comments: MASS STORAGE IS, MSI, CD  work good. Both MSI
20  ! and CD convert to MASS STORAGE IS.
30    ON ERROR CALL Testerror
40    CLEAR SCREEN
50    PRINT "*** CD or MSI TEST ****"
60    PRINT "Current directory is: ";SYSTEM$("MSI")
70    PRINT "Changing directory to C:\"
80    MASS STORAGE IS "C:\"
90    PRINT "MSI: ",SYSTEM$("MSI")
100   PRINT "Changing directory to D:\"
110   MASS STORAGE IS "D:\"
120   PRINT "MSI: ",SYSTEM$("MSI")
130   PRINT "Test passed."
140   END
150   SUB Testerror         !  Error Handler for
160     PRINT "* Error: ";ERRN      !  unexpected test program errors.
170     STOP
180   SUBEND
```

CHR$ EXAMPLE

```
10    CLEAR SCREEN
20    DISPLAY FUNCTIONS ON  !Make sure you can see all the characters.
30    FOR Loop=0 TO 255
40      DISP Loop;
50      PRINT CHR$(Loop);   !Display all the different characters.
60    NEXT Loop
70    PRINT CHR$(128)
80    PRINT CHR$(136)
90    END
```

CHRX EXAMPLE

```
10    PRINT CHRX,CHRY,RATIO   !Width and height of the characters.
20    END                     !RATIO returns the ratio of the x-y axis.
```

CHRY EXAMPLE

```
10    PRINT CHRX,CHRY,RATIO   !Returns the width and height of the character
font used.
20    END                     !RATIO returns the ratio of the x-y axis.
```

CINT EXAMPLE

```
10    DATA 2.6, 2.2, -2.2, -2.6
20    RESTORE
30    CLEAR SCREEN
40    PRINT "Variable value conversion test"
50    PRINT "Value","CINT(X)","FIX(X)","INT(X)"
60    FOR L=1 TO 4
70      READ X
80      PRINT X,CINT(X),FIX(X),INT(X)
90    NEXT L
100   END
```

CLEAR EXAMPLE

```
10  ! Use bus analyzer to verify commands; response is device specific
20    RESET 7
30    PAUSE
40    CLEAR 720
50    END
```

CLEAR ERROR EXAMPLE

```
10    CLEAR SCREEN
20    ON ERROR RECOVER Here
30    Force_error
40 Here: !
50    PRINT "The error received was:"
60    PRINT ERRM$
70    CLEAR ERROR
80    PRINT "Error Cleared"
90    PRINT "If another error message appears, the test failed."
100  END
110
120  SUB Force_error
130    PRINT SYSTEM$("HTBasic")!gives error 401
140  SUBEND
```

CLEAR LINE EXAMPLE

```
10    OUTPUT KBD;"I can clear this line with CLEAR LINE.";
20    WAIT 3
30    CLEAR LINE  !This clears the input line.
40    END
```

CLEAR SCREEN EXAMPLE

```
10    PRINT "I can clear the screen."
20    AREA PEN 6
30    MOVE 30,30
40    RECTANGLE 20,20,FILL
50    MOVE 50,50
60    AREA PEN 4
70    POLYGON 20,FILL
80    WAIT 2
90    CLEAR SCREEN
100   END
```

CLIP EXAMPLE

```
10    GINIT
20    CLEAR SCREEN
30    AREA PEN 4
40    CLIP 10,25,5,15
50    RECTANGLE 40,40,FILL
60    WAIT 2
70    PRINT "Now we will turn off the clip"
80    WAIT 2
90    CLIP OFF
100   RECTANGLE 40,40,FILL
110   END
```

CLS EXAMPLE

```
10    GINIT
20    CLEAR SCREEN
30    AREA PEN 4
40    CLIP 10,25,5,15
50    RECTANGLE 40,40,FILL
60    WAIT 2
70    PRINT "Now we will turn off the clip"
80    WAIT 2
90    CLIP OFF
100   RECTANGLE 40,40,FILL
110   END
```

CMPLX EXAMPLE

```
10    CLEAR SCREEN
20    COMPLEX A,B(1:10)
30    A=CMPLX(2,1)    !Makes a complex number 2+i.
40    FOR X=1 TO 10
50      B(X)=CMPLX(X,X/(-4))  !Makes various complex numbers depending on
loop.
60      PRINT B(X)
70    NEXT X
80    PRINT A
90    END
```

COLOR EXAMPLE

```
10    GINIT
20    PLOTTER IS CRT,"INTERNAL";COLOR MAP
30    AREA COLOR .8,.8,.8      !Change the FILL color
40    RECTANGLE 45,45,FILL,EDGE
50    MOVE 30,30
60    AREA COLOR .2,.9,.5      !Change the FILL color
70    RECTANGLE 45,45,FILL,EDGE
80    END
```

COM EXAMPLE

```
10    COM A$[20],Number
20    COM /Test/B$[40],Counter
30    A$="Hello"
40    B$="World"
50    Number=7
60    Counter=78
70    CLEAR SCREEN
80    PRINT "In main"
90    PRINT "A$=";A$,"Number=";Number
100   PRINT "B$=";B$,"Counter=";Counter
110   Subit
111   PRINT
120   PRINT "If SUB and MAIN variables are equal, then test passes."
150   END
160   SUB Subit
170     COM A$,Number
180     COM /Test/B$,Counter
181     PRINT
190     PRINT "In SUB"
200     PRINT "A$=";A$,"Number=";Number
210     PRINT "B$=";B$,"Counter=";Counter
220     Number=2
230   SUBEND
```

COMMAND$ EXAMPLE

```
10    CLEAR SCREEN
20    PRINT COMMAND$
30    END
```

COMPLEX EXAMPLE

```
10    CLEAR SCREEN
20    COMPLEX A,B(1:10)
30    A=CMPLX(2,1)    !Makes a complex number 2+i.
40    FOR X=1 TO 10
50      B(X)=CMPLX(X,X/(-4))  !Makes various complex numbers depending on
loop.
60      PRINT B(X)
70    NEXT X
80    PRINT A
90    END
```

CONFIGURE BDAT EXAMPLE

```
10    CONFIGURE BDAT MSB FIRST ! Will create bdat files for HP BASIC on
Workstation
20    CREATE BDAT "test.bdt",20
30    CAT "test.bdt"
40    PURGE "test.bdt"
40    END
```

CONFIGURE CREATE EXAMPLE

```
10    CONFIGURE CREATE "HTB"  !This sets the file being created to an HTB
header.
20    CREATE "test1.txt",0
30    CONFIGURE CREATE "HP"   !This sets the file being created to an HP
header.
40    CREATE "test2.txt",0
50    PURGE "test1.txt"
60    PURGE "test2.txt"
70    END
```

CONFIGURE DIM EXAMPLE

```
10    !Before you run this program,
20    !go to a command line and type
30    !the command CONFIGURE DIM OFF.
40    !If you don't, it won't work properly.
50    !The test should produce ERROR 16.
60    !Error 16 is produced because the
70    !variable A$ was not dimensioned first.
80    !With CONFIGURE DIM ON, you do not
90    !need to dimension a variable first.
100   ON ERROR GOTO 70
110   A$="Hello"
120   END
```

CONFIGURE DUMP EXAMPLE

```
10    CONFIGURE DUMP TO "WIN-DUMP"  !Use windows print driver for dumps to the
printer.
20    DUMP DEVICE IS 10             !Use windows default printer for dumps.
30    CALL Text
40    PRINT "Hello"
50    DUMP ALPHA
60    WAIT 2
70    CALL Graphic
80    DUMP GRAPHICS #10
90    END
100   SUB Text
110     CLEAR SCREEN
120     PRINT "*** Print TEST ****"
130     PRINT "[TAB]";TAB(15);"15 spaces"
140     PRINT TABXY(5,5);"Cool, gotoxy()"
150     PRINT "It is all good!!!"
160   SUBEND
170   SUB Graphic
180     CLEAR SCREEN
190     GINIT
200     MOVE 40,40
210     RECTANGLE 10,20
220     DISP "1"
230     AREA PEN 3
240     RECTANGLE 10,-20,FILL
250     DISP "2"
260     PEN 2
270     RECTANGLE -10,-20,EDGE
280     DISP "3"
290     AREA PEN 7
300     PEN 8
310     RECTANGLE -10,20,FILL,EDGE
320     DISP "4"
330     DISP
340   SUBEND
```

CONFIGURE KBD EXAMPLE

```
10    CONTROL KBD,203;1   !Turn off detection of mouse movements.
20    CLEAR SCREEN
30    PRINT "Mapping key '4' to '7'."
40    DISP "Press 'e' to stop"
50    Nustr$="0123756789"
60    Normstr$="0123456789"
70    PRINT "Type 4 and notice that it is now 7."
80    CONFIGURE KBD 48 TO Nustr$  !Change 4 to 7.
90    ON KBD GOTO Here
100   REPEAT
110 Loop: GOTO Loop
120 Here: Ch$=KBD$
130     PRINT Ch$
140   UNTIL Ch$="e"
150   CONFIGURE KBD 48 TO Normstr$     !Change 4 back to 4.
160   PRINT "Now 4 is back to normal."
170   DISP "All done."
180   END
```

CONFIGURE KEY EXAMPLE

```
10    CLEAR SCREEN
20    CONFIGURE KEY 51 TO NUM("<")  ! Changes key "3" to a left-arrow.
30    PRINT "Configuring key 3 to a left-arrow."
40    PRINT "Verify that this is true."
50    PRINT "Press continue to quit."
60    PAUSE
70    END
```

CONFIGURE LABEL EXAMPLE

```
10  ! This program configures the letter H to a "u" with two dots over it when
using LABEL.
20    CONFIGURE LABEL 72 TO
CHR$(134)&CHR$(10)&CHR$(218)&CHR$(86)&CHR$(69)&CHR$(21)&CHR$(6)&CHR$(140)&CHR$
(14)&CHR$(220)&CHR$(94)
30    MOVE 50,50
40    LABEL "Hello"  !The H in Hello should be a "u" with two dots over it.
50    END
```

CONFIGURE LONGFILENAMES EXAMPLE

```
10    CLEAR SCREEN
20    CREATE "CONFIGUREA234567890",1
30    CREATE
"CONFIGUREA234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890",1
40    CONFIGURE LONGFILENAMES ON
50    DISP "Long filename format. Program paused.."
60    CAT "configure*"
70    PAUSE
80    CLEAR SCREEN
90    DISP "Short filename format. Program paused.."
100  CONFIGURE LONGFILENAMES OFF
110  CAT "config*"
111  PAUSE
120  PURGE "CONFIGUREA234567890"
130  PURGE
"CONFIGUREA234567890A234567890A234567890A234567890A234567890A234567890A2345678
90A234567890A234567890"
140  CONFIGURE LONGFILENAMES ON
150  END
```

CONFIGURE MSI EXAMPLE

```
10    CONFIGURE MSI ON
20    CONFIGURE MSI ":,720,1" TO "c:\"
30    MASS STORAGE IS ":,720,1"
50    PRINT SYSTEM$("MSI")
60    CONFIGURE MSI OFF
80    PRINT SYSTEM$("MSI")
90    END
```

CONFIGURE PRT EXAMPLE

```
10    CONFIGURE PRT TO CRT  !Change PRT from a printer to the screen.
20    OUTPUT PRT;"Hello"
30    CONFIGURE PRT TO 10   !Change PRT back to a printer.
40    END
```

CONFIGURE SAVE EXAMPLE

```
10    CONFIGURE SAVE ASCII ON
20    CLEAR SCREEN
30    PRINT "SAVE test"
40    SAVE "save.txt"            !With CONFIGURE SAVE ASCII ON, notepad will
read the file as garbage.
50    EXECUTE "notepad save.txt"
60    PRINT "Program paused."
70    PURGE "save.txt"
80    PRINT "All done"
90    CONFIGURE SAVE ASCII OFF   !With CONFIGURE SAVE ASCII OFF, notepad will
read the file as normal text.
100   CLEAR SCREEN
110   PRINT "SAVE test"
120   SAVE "save.txt"
130   EXECUTE "notepad save.txt"
140   PRINT "Program paused."
150   PURGE "save.txt"
160   PRINT "All done"
170   END
```

CONJG EXAMPLE

```
10    CLEAR SCREEN
20    COMPLEX A,B,Y
30    A=CMPLX(2,1)
40    B=CONJG(A)   !Returns the conjugate of A.
50    PRINT "a = 2 + i :";A
60    PRINT "b = conjugate of a: ";B
70    END
```

CONTROL EXAMPLE

```
10    !LOAD BIN "SERIAL32"
20    CONTROL 9,3;19200  !Set the Serial baud rate to 19200.
30    PRINT STATUS(9,3)  !Make sure it was set correctly.
40    END
```

COPY EXAMPLE

```
10    Filename$="JUNK.XXX"
20    CLEAR SCREEN
30    SAVE Filename$
40    COPY Filename$ TO "JUNK.jnk"
50    DISP "Copied "&Filename$&" to JUNK.jnk - program paused"
60    CAT "Junk.*"
70    PAUSE
80    PURGE Filename$
90    PURGE "JUNK.jnk"
100   PRINT "Junk files were purged."
110   END
```

COS EXAMPLE

```
10     !    This example demonstrates the usage of the trigonometric
20     ! functions. The following triangle will be used:
30
40     !    |\
50     !    | a\   Given C = 4 units and angle
60     !   C|    \B            c = 53.1301023542  degrees
70     !    |b     c\           Note: angle b = 90 degrees.
80     !    +--------
90     !        A
100   CLEAR SCREEN
110   DEG ! get in degree mode
120   REAL A,B,C
130    ! Given:
140   C=4
150   Angle_c=53.1301023542
160   Angle_b=90
170    ! Angle a can be found by simply subtracting the total given
180    ! angles by 180 degrees. Every triangle only has 180
190    ! degress.
200   Angle_a=180-(Angle_c+Angle_b)
210    ! The sine of angle c is defined as C over B. Solving for
220    ! B gives us:
230   B=C/SIN(Angle_c)
240    ! The cosine of angle c is defined as A over B. Solving for
250    ! A gives us:
260   A=B*COS(Angle_c)
270    ! To double check the answers, one possible way is:
280    ! Given: A^2 + C^2 = B^2 and solving for C
290   IF SQR(B^2-A^2)=C THEN
300     PRINT "The leg A =";A;"units."
310     PRINT "The leg B =";B;"units."
320     PRINT "The leg C =";C;"units."
330     PRINT "Angle a is = ";Angle_a;"degrees."
340     PRINT "Angle b is = ";Angle_b;"degrees."
350     PRINT "Angle c is = ";Angle_c;"degrees."
360   ELSE
370     PRINT "An error has occurred."
380   END IF
390   END
```

COSH EXAMPLE

```
10    COMPLEX C
20    C=CMPLX(4,7)
30    CLEAR SCREEN
40    PRINT COSH(80)
50    X=REAL(COSH(C))
60    Y=COSH(REAL(C))*COS(IMAG(C))
70    PRINT X,Y
80    IF X=Y THEN
90      PRINT "True"
100   ELSE
110     PRINT "False - error in documentation."
120   END IF
130   X=IMAG(COSH(C))
140   Y=SINH(REAL(C))*SIN(IMAG(C))
150   PRINT X,Y
160   IF X=Y THEN
170     PRINT "True"
180   ELSE
190     PRINT "False - error in documentation."
200   END IF
210   END
```

COUNT EXAMPLE

```
10    Directory$="C:\"
20    CAT Directory$;NAMES,COUNT Count
30    PRINT "The number of files in "&Directory$&" is";Count
40    END
```

CREATE EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** output TEST ****"
30    CREATE "test.txt",0
40    ASSIGN @File TO "test.txt";FORMAT ON
50    A$="This is a test."
60    OUTPUT @File;A$
70    RESET @File
80    ENTER @File;Test$
90    IF A$=Test$ THEN
100     PRINT "Test passed."
110   ELSE
120     PRINT "Test failed. Output string did not equal input string."
130   END IF
140   ASSIGN @File TO *
150   PURGE "test.txt"
160   END
```

CREATE ASCII EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** output TEST ****"
30    CREATE ASCII "test.txt",0
40    ASSIGN @File TO "test.txt";FORMAT ON
50    A$="This is a test."
60    OUTPUT @File;A$
70    RESET @File
80    ENTER @File;Test$
90    IF A$=Test$ THEN
100     PRINT "Test passed."
110   ELSE
120     PRINT "Test failed. Output string did not equal input string."
130   END IF
140   ASSIGN @File TO *
150   PURGE "test.txt"
160   END
```

CREATE BDAT EXAMPLE

```
10    CLEAR SCREEN
20    CREATE BDAT "stuff1.XXX",67
30    CREATE BDAT "stuff2.XXX",67,78
40    CAT "*.XXX"
50    PRINT "CONT to purge files..."
60    PAUSE
70    PURGE "stuff1.XXX"
80    PURGE "stuff2.XXX"
90    PRINT "Files purged."
100   END
```

CREATE DIR EXAMPLE

```
10    CLEAR SCREEN
20    CONFIGURE LONGFILENAMES ON
30    CREATE DIR "New directory"    !Create the directory.
40    PRINT "I created a directory called"
50    CAT "New*";NAMES    !Make sure the directory was created.
60    PAUSE
70    PURGE "New directory"
80    PRINT "I removed it, now."
90    END
```

CRT EXAMPLE

```
10    CLEAR SCREEN
20    PRINT CRT
30    PRINTER IS CRT
40    PRINT "Hello"
50    END
```

CSIZE EXAMPLE

```
10    CLEAR SCREEN
20    FOR Height=1 TO 20
30      MOVE 0,40
40      DISP Height
50      CSIZE Height,1       !Change the size of the label font.
60      LABEL "Hello World!"
70      WAIT 1
80      CLEAR SCREEN
90    NEXT Height
100   END
```

CSUM EXAMPLE

```
10    OPTION BASE 1
20    DIM Matrix(3,3)
30    DIM Vector(3)
40    DATA 1, 2, 3, 4, 5, 6, 7, 8, 9
50    RESTORE
60    READ Matrix(*)
70    CLEAR SCREEN
80    PRINT "The matrix looks like: "
90    Prtmat(Matrix(*),3,3)
100   MAT Vector=CSUM(Matrix)
110   PRINT "The sum of each column is as follows"
120   PRINT Vector(*)
130   END
140   SUB Prtmat(A(*),Lenarr,Widarr)
150! This sub prints out a matrix length of Lenarr and wide as widarr.
160! A 3x3 matrix would print like:
170!           [ 1 2 3 ]  Widarr = 3
180!           [ 4 5 6 ]
190!           [ 7 8 9 ]
200!           Lenarr = 3
210     ASSIGN @Out TO CRT
220     FOR Col=1 TO Lenarr
230       OUTPUT @Out;"  [";
240       FOR Row=1 TO Widarr
250         OUTPUT @Out;A(Col,Row);
260       NEXT Row
270       OUTPUT @Out;" ]"
280     NEXT Col
290     ASSIGN @Out TO *
300   SUBEND
```

CYCLE EXAMPLE

```
10    ON CYCLE 5 GOTO Here
20    CLEAR SCREEN
30    PRINT "Loop until It is time."
40    LOOP
50      PRINT "Still waiting"
60      WAIT .5
70    END LOOP
80 Here: PRINT "It worked!"
90    PRINT "On cycle interrupts after 5 seconds."
100   END
```

DATA EXAMPLE

```
10    DIM Array(4)
20    DATA 1, 2, 3, 4, 5, "Hello user"
30    RESTORE   !Set pointer back to beginning of DATA statement.
40    CLEAR SCREEN
50    PRINT "Reading the data."
60    READ Array(*)   !Read in the numerics from DATA.
70    PRINT "The array: "
80    PRINT Array(*)
90    READ Str$       !Read in the string from DATA.
100   PRINT "and the string was:",Str$
110   END
```

DATE EXAMPLE

```
10    CLEAR SCREEN
20    PRINT DATE$(TIMEDATE)
30    PRINT "The number in seconds from 4713 B.C. is",FNJd(DATE$(TIMEDATE))
40    END
50    DEF FNJd(A$)
60      RETURN (DATE(A$) DIV 86400)-1
70    FNEND
```

DATE$ EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "The date today is ";DATE$(TIMEDATE)
30    PRINT "The number of seconds from 4713 B.C. is",FNJd(DATE$(TIMEDATE))
40    END
50    DEF FNJd(A$)
60      RETURN (DATE(A$) DIV 86400)-1
70    FNEND
```

DEALLOCATE EXAMPLE

```
10    ! *****************************************
20    !
30    ! TEST STACK MANIPULATIONS FOR ALLOCATE #4
40    !
50    ! *****************************************
60    CLEAR SCREEN
70    PRINT "Total Memory "&SYSTEM$("AVAILABLE MEMORY")
80    ON ERROR GOTO L140
90    ALLOCATE A$[32000],B(1000),INTEGER C(10)
100   PRINT "Memory after allocation "&SYSTEM$("AVAILABLE MEMORY")
110   DEALLOCATE A$,C(*),B(*)
120   PRINT "Memory freed "&SYSTEM$("AVAILABLE MEMORY")
130   ALLOCATE A$[32000],B(1000)
140   PRINT "Memory allocated again without the integer array "&SYSTEM$
("AVAILABLE MEMORY")
150   PRINT
160   PRINT "Memory before SUB call "&SYSTEM$("AVAILABLE MEMORY")
170   PRINT "Calling SUB..."
180   Yahoo
190   PRINT "Memory after SUB call "&SYSTEM$("AVAILABLE MEMORY")
200   PRINT
210   ALLOCATE Str$[90]
220   PRINT "Memory after allocated string "&SYSTEM$("AVAILABLE MEMORY")
230   ALLOCATE REAL D(8)
240   PRINT "Memory after allocated REAL array "&SYSTEM$("AVAILABLE MEMORY")
250   STOP
260 L140: PRINT "Test failed"
270   END
280     !
290   SUB Yahoo
300     PRINT "Memory in SUB call "&SYSTEM$("AVAILABLE MEMORY")
310     ALLOCATE COMPLEX Y(4)
320     PRINT "Memory after allocation of COMPLEX array "&SYSTEM$("AVAILABLE
MEMORY")
330   SUBEND
```

DEF FN EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Calling Add function to do the following:"
30    PRINT "5 + 8 =";FNAdd(5,8)
40    PRINT "Calling Message function."
50    PRINT FNMessage$("Hello")
60    END
70    DEF FNAdd(A,B)
80      RETURN A+B
90    FNEND
100   DEF FNMessage$(OPTIONAL String$)
110     RETURN String$&" was passed as a parameter."
120   FNEND
```

DEG EXAMPLE

```
10    Angle=80   !Degrees
20    Mode=1
30! DEG (degree) mode = 1
40! RAD (radian) mode = 0
50    CLEAR SCREEN
60    IF Mode THEN
70      DEG
80      PRINT "The sine of ";Angle;"degress is: ";SIN(Angle)
90    ELSE
100     RAD
110     PRINT "The sine of";Angle*(PI/180);"radians is: ";SIN(Angle)
120   END IF
130   END
```

DELAY EXAMPLE

```
10    CLEAR SCREEN
20    ON DELAY 3 GOTO Here
30    PRINT "I will wait 3 seconds, sit back and watch."
40 Loop: GOTO Loop
50    PRINT "Go on!"
60 Here: PRINT "Alright, I waited 3 seconds."
70    END
```

DELSUB EXAMPLE

```
10!   This program deletes an important SUB,
20! DO NOT save after you run!!!!!
30!   Notice in the code SUB's One and Two. SUB One
40! will be deleted with the DELSUB command.
50!   After you read the instructions comment out the
60! STOP statement.
70    CLEAR SCREEN
80    PRINT "Please read the instructions."
90    STOP
100   One
110   Two
120   DELSUB One
130   PRINT "SUB One deleted"
140   PRINT "Now, exit and do not save the file!"
150   END
160   SUB One
170     PRINT "Hello, in Sub one"
180   SUBEND
190   SUB Two
200     PRINT "Hello, in Sub two"
210   SUBEND
```

DET EXAMPLE

```
10    DIM Matrix(1:3,1:3)
20    DATA 1, 2, 3, 4, -5, 6, 7, 8, 9
30    RESTORE
40    CLEAR SCREEN
50    READ Matrix(*)
60    PRINT "The matrix looks like: "
70    Prtmat(Matrix(*),3,3)   !Print out the matrix used.
80    PRINT "The determinant is: ";DET(Matrix)
90    END
100   SUB Prtmat(A(*),Lenarr,Widarr)
110! This sub prints out a matrix length of Lenarr and wide as widarr.
120! A 3x3 matrix would print like:
130!          [ 1 2 3 ]  Widarr = 3
140!          [ 4 5 6 ]
150!          [ 7 8 9 ]
160!          Lenarr = 3
170     ASSIGN @Out TO CRT
180     FOR Col=1 TO Lenarr
190       OUTPUT @Out;"  [";
200       FOR Row=1 TO Widarr
210         OUTPUT @Out;A(Col,Row);
220       NEXT Row
230       OUTPUT @Out;" ]"
240     NEXT Col
250     ASSIGN @Out TO *
260   SUBEND
```

DIGITIZE EXAMPLE

```
10    !Click the mouse button and it will show you where you are on the
screen.
20    CLEAR SCREEN
30    WHILE I<>45
40      DIGITIZE X,Y,Stat$
50      PRINT "Loop";I;"x:";X,"y:";Y
60      PRINT "Status: ";Stat$;" - length: ";LEN(Stat$)
70      I=I+1
80    END WHILE
90    END
```

DIM EXAMPLE

```
10    OPTION BASE 0
20    DIM B$(55,2)[25]
30    DIM A$(155,2)[25]
40    DIM C$(25,2)[25]
50    MAT B$=(“E”)
60    PRINT B$(*)
70    MAT C$=(“T”)
80    MAT A$=B$(1:25,*)
90    MAT A$(1:4,*)=C$(1:4,*)
100   PRINT A$(*)
110   END
```

DISABLE EXAMPLE

```
10    ON KEY 1 GOTO Here
20    SET KEY 1,"LIST"
30    DISP "Hit F1"
40    WHILE Count<>5
50      Count=Count+1
60      WAIT 1
70      IF Count=2 THEN DISABLE !Once count=2, then pressing F1 will do
nothing.
80    END WHILE
90 Here: PRINT "Reached Here! Count = ";Count
100   SET KEY 1,"EDIT"
110   END
```

DISABLE INTR EXAMPLE

```
10    CLEAR SCREEN
30    RESET 7
40    ENABLE INTR 7;2! RESPONDS TO SRQ
50    ON INTR 7,1 GOTO Intrr
60    ON DELAY 10 GOTO Stopp
70    LOOP
80      OUTPUT 720;"HELLO, you."
90      DISP Counter
100     IF Counter=100 THEN
110       DISABLE INTR 7
120     END IF
130     Counter=Counter+1
140   END LOOP
150   STOP
160 Stopp:!
170   PRINT "TIMED OUT"
180   STOP
190 Intrr:!
200   PRINT "INTERRUPTED"
210   OUTPUT 720;"stop"
220   END
```

DISP EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "look at the display line."
30    DISP "I can clear this line."
40    WAIT 3
50    DISP
60    END
```

DISPLAY FUNCITONS EXAMPLE

```
10    DISPLAY FUNCTIONS OFF
20    FOR I=1 TO 32
30    PRINT CHR$(I);I
40    NEXT I
50    DISPLAY FUNCTIONS ON
60    FOR I=1 TO 32
61    PRINT CHR$(I);I
62    NEXT I
70    END
```

DIV EXAMPLE

```
10    IF (8 DIV 3.0)=(FIX(8/3.0)) THEN PRINT "Good."
20    END
```

DOT EXAMPLE

```
10    DATA 1,2,3,4,5,6
20    RESTORE
30 Vector: IMAGE 3("[",DD,"]",/)   !Printing format.
40 Row_vector: IMAGE 3("[",DD,"]") !Printing format.
50    OPTION BASE 1
60    INTEGER A(3),B(3)
70    CLEAR SCREEN
80    READ A(*),B(*)
90    PRINT "Vector a = "
100   PRINT USING Row_vector;A(*)
110   PRINT "Vector b = "
120   PRINT USING Vector;B(*)
130   PRINT "The DOT product is ";DOT(A,B)
140   END
```

DRAW EXAMPLE

```
10    CLEAR SCREEN
20    GINIT
30    PRINT "Program pauses a lot, press CONT."
40    PEN 6
50    FOR Loop=1 TO 10
60      MOVE 10+(Loop*10),40
70      DRAW 10+(Loop*10),80
80      DISP "Line type: ";Loop
90      PAUSE
100     LINE TYPE Loop
110   NEXT Loop
120   END
```

DROUND EXAMPLE

```
10    Number=656576
20    CLEAR SCREEN
30    FOR Roundto=1 TO 6
40      PRINT Roundto,DROUND(Number,Roundto) !Round to significant digits
indicated by Roundto.
50    NEXT Roundto
60    END
```

DUMP EXAMPLE

```
10    CONFIGURE DUMP TO "WIN-DUMP" !Use the windows print driver for dumps.
20    DUMP DEVICE IS PRT           !Set dump device to windows default
printer.
30    Text
40    PRINT "Hello"
50    DUMP ALPHA                   !Dump out the text on the screen.
60    CLEAR SCREEN
70    PRINT "Press CONT"
80    PAUSE
90    Graphic
100   DUMP GRAPHICS #10            !Dump out the graphics on the screen.
110   END
120   SUB Text
130     CLEAR SCREEN
140     PRINT "*** Print TEST ****"
150     PRINT "[TAB]";TAB(15);"15 spaces"
160     PRINT TABXY(5,5);"TAB test"
170   SUBEND
180   SUB Graphic
190     CLEAR SCREEN
200     GINIT
210     MOVE 40,40
220     RECTANGLE 10,20
230     DISP "1"
240     AREA PEN 3
250     RECTANGLE 10,-20,FILL
260     DISP "2"
270     PEN 2
280     RECTANGLE -10,-20,EDGE
290     DISP "3"
300     AREA PEN 7
310     PEN 8
320     RECTANGLE -10,20,FILL,EDGE
330     DISP "4"
340     DISP
350   SUBEND
```

DUMP DEVICE IS EXAMPLE

```
10    CONFIGURE DUMP TO "WIN-DUMP" !Use the windows print driver for dumps.
20    DUMP DEVICE IS PRT           !Set dump device to windows default printer.
30    Text
40    PRINT "Hello"
50    DUMP ALPHA                   !Dump out the text on the screen.
60    CLEAR SCREEN
70    PRINT "Press CONT"
80    PAUSE
90    Graphic
100   DUMP GRAPHICS #10            !Dump out the graphics on the screen.
110   END
120   SUB Text
130     CLEAR SCREEN
140     PRINT "*** Print TEST ****"
150     PRINT "[TAB]";TAB(15);"15 spaces"
160     PRINT TABXY(5,5);"TAB test"
170   SUBEND
180   SUB Graphic
190     CLEAR SCREEN
200     GINIT
210     MOVE 40,40
220     RECTANGLE 10,20
230     DISP "1"
240     AREA PEN 3
250     RECTANGLE 10,-20,FILL
260     DISP "2"
270     PEN 2
280     RECTANGLE -10,-20,EDGE
290     DISP "3"
300     AREA PEN 7
310     PEN 8
320     RECTANGLE -10,20,FILL,EDGE
330     DISP "4"
340     DISP
350   SUBEND
```

DVAL EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "DVAL conversion test."
30    PRINT
40    PRINT DVAL("00000000000000000000000001100010",2)! From Binary to
Decimal.
50    PRINT DVAL("142",8)  ! From Octal to Decimal.
60    PRINT DVAL("98",10)  ! From base 10 to Decimal.
70    PRINT DVAL("62",16)  ! From Hex to Decimal.
80    PRINT
90    PRINT "See DVAL$.prg as well."
100   END
```

DVAL$ EXAMPLE

```
10    Number=98
20    CLEAR SCREEN
30    PRINT "DVAL$ conversion test."
40    PRINT
50    PRINT "Convert";Number;"to... "
60    PRINT "binary: ",DVAL$(Number,2)! to binary
70    PRINT "octal: ",DVAL$(Number,8)! to octal
80    PRINT "base ten: ",DVAL$(Number,10)! to base 10
90    PRINT "hex: ",DVAL$(Number,16)! to hex
100   END
```

ECHO EXAMPLE

```
10    PLOTTER IS CRT,"INTERNAL";COLOR MAP
20    CLEAR SCREEN
30    SET ECHO 65,50    !Set the crosshair to the middle of the screen.
40    READ LOCATOR X,Y  !Read where the mouse pointer is at.
50    PRINT X,Y
60    END
```

EDGE EXAMPLE

```
10    CLEAR SCREEN
20    GINIT
30    MOVE 40,40
40    RECTANGLE 10,20
50    DISP "1"
60    WAIT 1
70    AREA PEN 3
80    RECTANGLE 10,-20,FILL
90    DISP "2"
100   WAIT 1
110   PEN 2
120   RECTANGLE -10,-20,EDGE
130   DISP "3"
140   WAIT 1
150   AREA PEN 7
160   PEN 8
170   RECTANGLE -10,20,FILL,EDGE
180   DISP "4"
190   WAIT 1
200   DISP
210   END
```

ELSE EXAMPLE

```
10    IF NOT 1 THEN
20      CLEAR SCREEN
30      PRINT 5
40      BEEP
50    ELSE
60      PRINT "NOT 1"
70      STOP
80    END IF
90    PRINT "all good"
100   END
```

ENABLE INTR EXAMPLE

```
10    RESET 7
30    ENABLE INTR 7;2        ! RESPONDS TO SRQ
40    ON INTR 7,1 GOTO Intrr
50    ON TIMEOUT 7,30 GOTO Stopp
60    LOOP
70      OUTPUT 720;"HELLO"
80    END LOOP
90    STOP
100 Stopp:!
110   PRINT "TIMED OUT"
120   STOP
130 Intrr:!
140   PRINT "INTERRUPTED"
150   END
```

END EXAMPLE

```
10    CLEAR SCREEN
20    END
```

END IF EXAMPLE

```
10    IF 1 THEN
20      CLEAR SCREEN
30      PRINT 5
40      BEEP
50    ELSE
60      PRINT "NO"
70      STOP
80    END IF
90    PRINT "all good"
100   END
```

END LOOP EXAMPLE

```
10    CLEAR SCREEN
20    LOOP
30      PRINT "Iterarion:";Counter
40      PRINT Counter MOD 3
50      Counter=Counter+1
60    EXIT IF Counter=5 OR (Counter MOD 4)=3
70      PRINT "Not finished."
80    END LOOP
90    PRINT "All done."
100   END
```

END SELECT EXAMPLE

```
10    CLEAR SCREEN
20    INPUT "Please enter your age:",Age
30    SELECT Age
40    CASE <1,>100
50      PRINT "Congratulations - Movie is free!"
60      GOTO End
70    CASE <12
80      Price=2.00
90    CASE 12 TO 59
100     Price=6.50
110   CASE 60
120     PRINT "Special movie rate"
130     Price=3.00
140   CASE ELSE
150     Price=4.50
160   END SELECT
170 Image: IMAGE "Movie price is $", D.2D
180   PRINT USING Image;Price
190 End: END
```

END WHILE EXAMPLE

```
10    Good=6
20    WHILE Good
30      PRINT Good
40      Good=Good-1
50    END WHILE
60    END
```

ENTER EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** output TEST ****"
30    CREATE "test.txt",0
40    PRINT "Created file:"
50    CAT "*.txt";NAMES
60    ASSIGN @File TO "test.txt";FORMAT ON
70    OUTPUT @File;"Hello world."
80    PRINT "Wrote to file."
90    RESET @File
100   ENTER @File;Test$
110   PRINT "Read string from file: ";Test$
120   ASSIGN @File TO *
130   PRINT "CONT to purge file."
140   PAUSE
150   PURGE "test.txt"
160   PRINT "File purged."
170   END
```

ENVIRON$ EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Some environment variables are defined as:"
30    PRINT "PATH:"
40    PRINT ENVIRON$("PATH")
50    PRINT "TEMP:"
60    PRINT ENVIRON$("TEMP")
70    END
```

EOL EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** Assign Buffer TEST ****"
30    ASSIGN @Test TO 720;EOL OFF
40    ASSIGN @Out TO CRT
50    OUTPUT @Test;"Hello, how are you?"
60    PRINT "All done."
70    END
```

ERRL EXAMPLE

```
10    CLEAR SCREEN
20    ON ERROR GOTO 50
30   !force error
40 Here: PRINT SYSTEM$("NON EXISTAT ") !gives error 401
50    PRINT "Testing line label for error."
60    IF ERRL(Here) THEN CALL Testerr   !See if the error occurred on line 40.
70    PRINT "If you see this line, the test failed."
80    END
90    SUB Testerr
100     BEEP
110     PRINT "Error *";ERRN;"on line";ERRLN
120     STOP
130   SUBEND
```

ERRLN EXAMPLE

```
10    CLEAR SCREEN
20    ON ERROR CALL Testerr
30    Forceerr
40    PRINT "If you see this line, the test failed."
50    END
60    SUB Forceerr
70   !force error
80      PRINT SYSTEM$("NON EXISTAT")!gives error 401
90    SUBEND
100   SUB Testerr
110     BEEP
120     PRINT "Error *";ERRN;"on line";ERRLN
130     STOP
140   SUBEND
```

ERRM$ EXAMPLE

```
10    CLEAR SCREEN
20    ON ERROR CALL Testerr
30    Forceerr
40    PRINT "If you read this message, the test failed."
50    END
60    SUB Forceerr
70  !force error
80      PRINT SYSTEM$("NON EXISTAT ")!gives error 401
90    SUBEND
100   SUB Testerr
110     BEEP
120     PRINT "*";ERRM$
130     STOP
140   SUBEND
```

ERRN EXAMPLE

```
10    CLEAR SCREEN
20    ON ERROR CALL Testerr
30    Forceerr
40    PRINT "If you read this message, the test failed."
50    END
60    SUB Forceerr
70  !force error
80      PRINT SYSTEM$("NON EXISTAT ")  !gives error 401
90    SUBEND
100   SUB Testerr
110     BEEP
120     PRINT "Error *";ERRN;"on line";ERRLN
130     STOP
140   SUBEND
```

ERROR EXAMPLE

```
10    CLEAR SCREEN
20    ON ERROR CALL Testerr
30    Forceerr
40    PRINT "You should never see this line."
50    END
60    SUB Forceerr
70  !force error
80      PRINT SYSTEM$("NON EXISTAT ")   !gives error 401
90    SUBEND
100   SUB Testerr
110     BEEP
120     PRINT "Error *";ERRN;"on line";ERRLN
130     STOP
140   SUBEND
```

ERROR RETURN EXAMPLE

```
10    ON ERROR GOSUB Here
20    CLEAR SCREEN
30    CAUSE ERROR 0
40    PRINT "Skip error"
50    STOP
60 Here: PRINT "On";I;"- Intercepted error";ERRN
70    ERROR RETURN
80    PRINT "This line is never reached."
90    END
```

ERROR SUBEXIT EXAMPLE

```
10    ON ERROR CALL Here
20    CLEAR SCREEN
30    CAUSE ERROR 0
40    PRINT "Skip error, and continue..."
50    PRINT "test complete"
60    END
70    SUB Here
80      PRINT "On cause error ";I;"- Intercepted error";ERRN
90      ERROR SUBEXIT
100     PRINT "Should have exited before this statement."
110   SUBEND
```

EXECUTE EXAMPLE

```
10    DIM S$[40]
20    REPEAT
30      CLEAR SCREEN
40      PRINT "Hello, welcome to PC Pro"
50      PRINT
60      PRINT "   PC Pro is your pseudo DOS prompt."
70      PRINT "Type EXIT to quit."
80      INPUT "Enter command to run:",S$
90      IF S$<>"exit" THEN
100      EXECUTE S$
110    END IF
120  UNTIL S$="exit"
130  CLEAR SCREEN
140  PRINT "Good bye!"
150  END
```

EXIT IF EXAMPLE

```
10    CLEAR SCREEN
20    LOOP
30      PRINT "Iteration:";Counter
40      PRINT Counter MOD 3
50      Counter=Counter+1
60    EXIT IF Counter=5 OR (Counter MOD 4)=3
70      PRINT "Not finished."
80    END LOOP
90    PRINT "All done."
100   END
```

EXOR EXAMPLE

```
10  ! This program prints the truth table for an EXOR function.
20    DATA 0,0,0,1,1,0,1,1
30    RESTORE
40    CLEAR SCREEN
50    PRINT "EXOR test"
60    PRINT " J"," K","J EXOR K"
70    FOR L=1 TO 4
80      READ J,K
90      PRINT J,K,J EXOR K
100   NEXT L
110   END
```

EXP EXAMPLE

```
10    Begin_balance=2000
20    Rate=.08
30    Years=8
40    Balance=Begin_balance*EXP(Rate*Years)
50 Money: IMAGE "$", 5DD.DD
60 Percent: IMAGE "%", DD
70    CLEAR SCREEN
80    PRINT "By putting"
90    PRINT USING Money;Begin_balance
100   PRINT "in an account which is compounded continuously at"
110   Rate=Rate*100
120   PRINT USING Percent;Rate
130   PRINT "At the end of";Years;"years you will have"
140   PRINT USING Money;Balance
150   END
```

EXPANDED EXAMPLE

```
10    CONFIGURE DUMP TO "WIN-DUMP" !Use the windows print driver for dumps.
20    DUMP DEVICE IS PRT,EXPANDED  !Set dump device to windows default
printer.
30    Text                         !EXPANDED option rotates the picture 90
degrees on the page.
40    PRINT "Hello"
50    DUMP ALPHA                   !Dump out the text on the screen.
60    CLEAR SCREEN
70    PRINT "Press CONT"
80    PAUSE
90    Graphic
100   DUMP GRAPHICS #10            !Dump out the graphics on the screen.
110   END
120   SUB Text
130     CLEAR SCREEN
140     PRINT "*** Print TEST ****"
150     PRINT "[TAB]";TAB(15);"15 spaces"
160     PRINT TABXY(5,5);"TAB test"
170   SUBEND
180   SUB Graphic
190     CLEAR SCREEN
200     GINIT
210     MOVE 40,40
220     RECTANGLE 10,20
230     DISP "1"
240     AREA PEN 3
250     RECTANGLE 10,-20,FILL
260     DISP "2"
270     PEN 2
280     RECTANGLE -10,-20,EDGE
290     DISP "3"
300     AREA PEN 7
310     PEN 8
320     RECTANGLE -10,20,FILL,EDGE
330     DISP "4"
340     DISP
350   SUBEND
```

FILL EXAMPLE

```
10    CLEAR SCREEN
20    GINIT
30    MOVE 40,40
40    RECTANGLE 10,20
50    DISP "1"
60    WAIT 1
70    AREA PEN 3
80    RECTANGLE 10,-20,FILL
90    DISP "2"
100   WAIT 1
110   PEN 2
120   RECTANGLE -10,-20,EDGE
130   DISP "3"
140   WAIT 1
150   AREA PEN 7
160   PEN 8
170   RECTANGLE -10,20,FILL,EDGE
180   DISP "4"
190   WAIT 1
200   DISP
210   END
```

FIX EXAMPLE

```
10    DATA 2.6, 2.2, -2.2, -2.6
20    RESTORE
30    CLEAR SCREEN
40    PRINT "Variable value conversion test"
50    PRINT "Value","CINT(X)","FIX(X)","INT(X)"
60    FOR L=1 TO 4
70      READ X
80      PRINT X,CINT(X),FIX(X),INT(X)
90    NEXT L
100   END
```

FN EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "5 + 8 =";FNAdd(5,8)
30    PRINT FNMessage$("Hello")
40    END
50    DEF FNAdd(A,B)
60      RETURN A+B
70    FNEND
80    DEF FNMessage$(OPTIONAL String$)
90      IF NPAR=0 THEN RETURN "You didn't use the OPTIONAL parameter."
100     RETURN String$&" was a good choice."
110   FNEND
```

FNEND EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "5 + 8 =";FNAdd(5,8)
30    PRINT FNMessage$("Hello")
40    END
50    DEF FNAdd(A,B)
60      RETURN A+B
70    FNEND
80    DEF FNMessage$(OPTIONAL String$)
90      IF NPAR=0 THEN RETURN "You didn't use the OPTIONAL parameter."
100     RETURN String$&" was a good choice."
110   FNEND
```

FOR ... NEXT EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "For loop demo."
30    PRINT "Count from 40 to 500 by 20."
40    FOR J=40 TO 500 STEP 20
50      PRINT TAB(5),J
60      WAIT .25
70    NEXT J
80    PRINT "test complete"
90    END
```

FORMAT EXAMPLE

```
10    CLEAR SCREEN
20    CREATE "test.txt",0
30    PRINT "Created file: ";
40    CAT "*.txt";NAMES
50    ASSIGN @File TO "test.txt";FORMAT ON  !FORMAT ON means it's an ordinary
ASCII file.
60    OUTPUT @File;"This is a test"
70    PRINT "Output a string to the file."
80    RESET @File
90    PRINT "Reset the file to the beginning and read string."
100  ENTER @File;Test$
110  PRINT "The string read was: ";Test$
120  ASSIGN @File TO *
130  PURGE "test.txt"
140  PRINT "File purged."
150  END
```

FRE EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Amount of free memory: ";FRE
30    END
```

FROM EXAMPLE

```
10    CLEAR SCREEN
20    MASS STORAGE IS "d:"
30    READ LABEL Id$    !Reads the LABEL on the drive.
40    READ LABEL Id2$ FROM "c:"
50    PRINT SYSTEM$("MSI");Id$
60    PRINT "C:\ ";Id2$
70    END
```

GCLEAR EXAMPLE

```
10    RECTANGLE 20,40,FILL,EDGE
20    PRINT "In two seconds, I will GCLEAR"
30    WAIT 2
40    GCLEAR
50    END
```

GESCAPE EXAMPLE

```
10     CALL Code1
20     CALL Code2
30     CALL Code3
40     CALL Code4_5
50     CALL Code6
60     CALL Code102
70     CALL Code103
80   ! CALL Code104
90     CALL Code106
100    CALL Code130
110    CALL Code131
120    CALL Code132
130    CALL Code135
140    CALL Code138
150    CALL Code137
160    CALL Code141
170    CALL Code30
180    CALL Code31
190    CALL Code32
200    CALL Code33
210    CALL Code34
220    CALL Code35
230    CALL Code36
240    CALL Code37
250    CALL Code38
260    CALL Code39
270    CALL Code41
280    END
290    !
300    ! Gescape code 1 returns the number of color map entries.
310    ! A typical computer will return 256 as the answer.
320    !
330    SUB Code1
340      INTEGER A_return(0)
350      GESCAPE CRT,1;A_return(*)
360      PRINT "There are";A_return(0);"color map entries."
370      WAIT 2
380      CLEAR SCREEN
390    SUBEND
400    !
410    ! Gescape code 2 returns color map values. For example, the first
```

```
420   ! row of the array contains information for pen 0, the second
430   ! for pen 1, and so on. The first column of the array is the
440   ! red value, the second green, and the third blue.
450   !
460   SUB Code2
470     REAL B_return(15,2)
480     GESCAPE CRT,2;B_return(*)
490     PRINT "PEN","RED","GREEN","BLUE"
500     FOR I=0 TO 15
510       PRINT
I,DROUND(B_return(I,0),1),DROUND(B_return(I,1),1),DROUND(B_return(I,2),1)
520     NEXT I
530     WAIT 2
540     CLEAR SCREEN
550   SUBEND
560   !
570   ! Gescape code 3 returns the hard-clip values and GSTORE array size.
580   ! The return array must be a one dimensional INTEGER array and must
590   ! contain at least four elements. The first four elements of the array
600   ! are assigned the values Xmin, Ymin, Xmax, and Ymax. For a CRT, the
fifth
610   ! and sixth elements give the INTEGER array dimensions needed by the
GSTORE
620   ! command to store the screen image.
630   !
640   SUB Code3
650     INTEGER C_return(5)
660     GESCAPE CRT,3;C_return(*)
670     PRINT "Xmin","Ymin","Xmax","Ymax","Rows","Columns"
680     PRINT
C_return(0),C_return(1),C_return(2),C_return(3),C_return(4),C_return(5)
690     WAIT 2
700     CLEAR SCREEN
710   SUBEND
720   !
730   ! Gescape codes 4 and 5 change the graphics writing mode. If the code is
4,
740   ! the drawing mode is set to normal. If the code is 5, the drawing mode
750   ! is set to alternate. See the User's Guide for specific information.
760   !
770   SUB Code4_5
780     GESCAPE CRT,5  !Set to alternate drawing mode.
790     GESCAPE CRT,4  !Set to normal drawing mode.
800   SUBEND
```

```
810   !
820   ! Gescape code 6 returns the graphics display mask. The return array
must be a
830   ! one dimensional INTEGER array, and must have at least one element. The
first
840   ! element is assigned the value of the graphics write-enable mask. The
second
850   ! element, if present, is assigned the value of the graphics display
enable mask.
860   !
870   SUB Code6
880     INTEGER D_return(1)
890     GESCAPE CRT,6;D_return(*)
900     PRINT "Graphics write enable mask :";D_return(0)
910     PRINT "Graphics display enable mask :";D_return(1)
920     WAIT 2
930     CLEAR SCREEN
940   SUBEND
950   !
960   ! Gescape code 102 returns the current VIEWPORT and WINDOW values. The
return
970   ! array should be a two dimensional REAL array with two rows and four
columns.
980   !
990   SUB Code102
1000    REAL W(1,3)
1010    GESCAPE CRT,102;W(*)
1020    PRINT "The current window is  ";W(0,0),W(0,1),W(0,2),W(0,3)
1030    PRINT "The current viewport is";W(1,0),W(1,1),W(1,2),W(1,3)
1040    WAIT 2
1050    CLEAR SCREEN
1060  SUBEND
1070  !
1080  ! Gescape code 103 returns the current PEN and AREA PEN assignments. The
return
1090  ! array should be a one dimensional INTEGER array with two elements. The
first
1100  ! element is assigned the current PEN assignment. The second element is
assigned the
1110  ! current AREA PEN assignment.
1120  !
1130  SUB Code103
1140    INTEGER P(1)
1150    GESCAPE CRT,103;P(*)
1160    PRINT "The current      PEN is";P(0)
```

```
1170    PRINT "The current AREA PEN is";P(1)
1180    WAIT 2
1190    CLEAR SCREEN
1200  SUBEND
1210  !
1220  ! Gescape code 104 sets device-specific information. The param array
must be a one
1230  ! dimensional INTEGER array. The number of elements required depends on
the device
1240  ! driver. Conventionally, it contains two elements. The first element is
the operation
1250  ! number and the second element is the value associated with that
operation.
1260  !
1270  SUB Code104
1280    INTEGER Param(1)
1290    Param(0)=1     !HPGL Operation Number: 1 = HPGL/2 Flag
1300    Param(1)=1     !Value: 1=enable, 0=disable
1310    GESCAPE 7,104,Param(*) !7 is the ISC. You can use any ISC.
1320  SUBEND
1330  !
1340  ! Gescape code 105 sets device-specific information in the GRAPHICS
INPUT IS device.
1350  ! It is the same as the gescape code 104 shown above. The only
difference is that you
1360  ! are sending codes specific for the GRAPHICS INPUT IS device you are
using.
1370  !
1380  !
1390  ! Gescape code 106 sets device-specific information in the DUMP DEVICE
IS device.
1400  ! The param array must be a one dimensional INTEGER array. The number of
elements
1410  ! required depends on the device driver. The first element is the
operation
1420  ! number and the subsequent elements are the values associated with that
1430  ! operation.
1440  !
1450  SUB Code106
1460  PAUSE
1470    INTEGER A(1:5)
1480    CONTROL 26,102;2
1490    CONFIGURE DUMP TO "PCL"
1500    DUMP DEVICE IS 26
1510    A(1)=1   ! operation code, always 1
```

```
1520    A(2)=100 ! begin row, screen units
1530    A(3)=300 ! end row, screen units
1540    A(4)=0   ! reserved, must be 0
1550    A(5)=0   ! reserved, must be 0
1560    GESCAPE 26,106,A(*)
1570    FRAME
1580    MOVE 0,0
1590    DRAW 100,100
1600    DUMP GRAPHICS
1610  SUBEND
1620  !
1630  ! Gescape code 130 maximizes the Basic child window.
1640  !
1650  SUB Code130
1660    GESCAPE CRT,130
1670    WAIT 2
1680  SUBEND
1690  !
1700  ! Gescape code 131 hides the Basic child window.
1710  !
1720  SUB Code131
1730    PRINT "The Basic child window will now be hidden."
1740    WAIT 2
1750    GESCAPE CRT,131
1760    WAIT 2
1770  SUBEND
1780  !
1790  ! Gescape code 132 restores the Basic child window once it has been
hidden.
1800  !
1810  SUB Code132
1820    GESCAPE CRT,132
1830    PRINT "The Basic child window has been restored."
1840    WAIT 2
1850  SUBEND
1860  !
1870  ! Gescape code 135 brings the Basic child window to the top.
1880  !
1890  SUB Code135
1900    GESCAPE CRT,135
1910    WAIT 2
1920  SUBEND
1930  !
```

```
1940  ! Gescape code 138 Hides/Restores the title bar of the child window.
1950  !
1960  SUB Code138
1970    GESCAPE CRT,138
1980    PRINT "The title bar has been hidden."
1990    WAIT 2
2000    GESCAPE CRT,138
2010    PRINT "The title bar has been restored."
2020    CLEAR SCREEN
2030  SUBEND
2040  !
2050  ! Gescape 137 returns the Title Bar enable flag of the child window.
2060  !
2070  SUB Code137
2080    INTEGER X(0)
2090    GESCAPE CRT,138
2100    GESCAPE CRT,137;X(*)
2110    DISP "The Title Bar enable flag is";X(*);"with the Title Bar hidden."
2120    GESCAPE CRT,138
2130    WAIT 2
2140    GESCAPE CRT,137;X(*)
2150    PRINT "The Title Bar enable flag is";X(*);"with the Title Bar
restored."
2160    WAIT 2
2170    CLEAR SCREEN
2180  SUBEND
2190  !
2200  ! Gescape code 141 minimizes the Basic child window.
2210  !
2220  SUB Code141
2230    GESCAPE CRT,141
2240    WAIT 2
2250    GESCAPE CRT,130
2260  SUBEND
2270  !
2280  ! Gescape code 30 maximizes the Basic parent window.
2290  !
2300  SUB Code30
2310    GESCAPE CRT,30
2320    PRINT "The parent window is maximized."
2330    WAIT 2
2340    CLEAR SCREEN
2350  SUBEND
```

```
2360  !
2370  ! Gescape code 31 hides the Basic parent window.
2380  !
2390  SUB Code31
2400    PRINT "The parent window will now be hidden."
2410    WAIT 2
2420    GESCAPE CRT,31
2430    WAIT 2
2440  SUBEND
2450  !
2460  ! Gescape code 32 restores the Basic parent window once it has been
hidden.
2470  !
2480  SUB Code32
2490    GESCAPE CRT,32
2500    PRINT "The parent window has been restored."
2510    WAIT 2
2520    CLEAR SCREEN
2530  SUBEND
2540  !
2550  ! Gescape code 33 is used to set the parent window position and size.
2560  !
2570  SUB Code33
2580    INTEGER Set(1:4)
2590    DATA 90,100,500,300
2600    READ Set(*)
2610    GESCAPE CRT,33,Set(*)
2620    PRINT "The parent window is now at position";Set(1);",";Set(2)
2630    PRINT "Its width is";Set(3);"and its height is";Set(4)
2640    WAIT 2
2650    GESCAPE CRT,41
2660    GESCAPE CRT,30
2670    WAIT 2
2680    CLEAR SCREEN
2690  SUBEND
2700  !
2710  ! Gescape code 34 gets the parent window position and size.
2720  !
2730  SUB Code34
2740    INTEGER Get(1:4)
2750    GESCAPE CRT,34;Get(*)
2760    PRINT "The parent window is located at";Get(1);",";Get(2)
2770    PRINT "Its width is";Get(3);"and its height is";Get(4)
```

```
2780    WAIT 2
2790    CLEAR SCREEN
2800  SUBEND
2810  !
2820  ! Gescape code 35 brings the parent window back to the top of the
screen.
2830  !
2840  SUB Code35
2850    GESCAPE CRT,35
2860    WAIT 2
2870  SUBEND
2880  !
2890  ! Gescape code 36 gets the screen size.
2900  !
2910  SUB Code36
2920    INTEGER G(1:2)
2930    GESCAPE CRT,36;G(*)
2940    PRINT "The screen dimensions are";G(1);"by";G(2)
2950    WAIT 2
2960    CLEAR SCREEN
2970  SUBEND
2980  !
2990  ! Gescape code 37 returns the Title Bar enable flag of the parent
window.
3000  !
3010  SUB Code37
3020    INTEGER X(0)
3030    GESCAPE CRT,38
3040    GESCAPE CRT,37;X(*)
3050    PRINT "The Title Bar enable flag is";X(*);"with the Title Bar hidden."
3060    WAIT 2
3070    GESCAPE CRT,38
3080    GESCAPE CRT,37;X(*)
3090    PRINT "The Title Bar enable flag is";X(*);"with the Title Bar
restored."
3100    WAIT 2
3110    CLEAR SCREEN
3120  SUBEND
3130  !
3140  ! Gescape code 38 Hides/Restores the Title Bar of the parent window.
3150  !
3160  SUB Code38
3170    GESCAPE CRT,38
```

```
3180    PRINT "The title bar has been hidden."
3190    WAIT 2
3200    GESCAPE CRT,38
3210    PRINT "The title bar has been restored."
3220    WAIT 2
3230    CLEAR SCREEN
3240  SUBEND
3250  !
3260  ! Gescape code 39 sets the DUMP size (% of paper width).
3270  !
3280  SUB Code39
3290    INTEGER S(1:1)
3300    S(1)=50
3310    GESCAPE CRT,39,S(*)
3320    DUMP DEVICE IS 10
3330    CONFIGURE DUMP TO "WIN-DUMP"
3340    DUMP GRAPHICS
3350  SUBEND
3360  !
3370  ! Gescape code 41 minimizes the parent window.
3380  !
3390  SUB Code41
3400    GESCAPE CRT,41
3410    WAIT 2
3420    GESCAPE CRT,32
3430  SUBEND
```

GET EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "CONT to GET file."
30    PAUSE
40    GET "print.prg"
50    END
```

GFONT IS EXAMPLE

```
10   MOVE 10,50
11   GFONT IS ""
20   LABEL "Default label text"
30   GFONT IS "courier"
40   LABEL "label text after GFONT IS ""COURIER"""
50   END
```

GINIT EXAMPLE

```
10    MERGE ALPHA WITH GRAPHICS
20    PRINT TABXY(30,13);"hello"
30    AREA PEN 2
40    MOVE 50,50
50    POLYGON 30,FILL
60    WAIT 2
70    GINIT
80    MOVE 50,50
90    PEN 5
100   POLYGON 20
110   END
```

GLOAD EXAMPLE

```
10    INTEGER A(1:6)
20    GESCAPE CRT,3;A(*)
30    ALLOCATE INTEGER B(1:A(5),1:A(6))
40    GINIT
50    GCLEAR
60    CLEAR SCREEN
70    MOVE 40,40
80    AREA PEN 7
90    RECTANGLE 20,20,FILL
100   GSTORE B(*)
110   WAIT 3
120   CLEAR SCREEN
130   MOVE 30,40
140   GLOAD B(*)
150   END
```

GOSUB EXAMPLE

```
10    Y=3
20    Z=4
30    GOSUB Calc_x
40    PRINT "X = ";X
50    STOP
60 Calc_x: X=Y*45/Z
70    RETURN
80    END
```

GOTO EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "This is the wacky GOTO example."
30    GOTO L5
40 L1: PRINT "Now, I am at line 1 (L1)."
50    PRINT "L1 - GOTO L3"
60    GOTO L3
70 L2: PRINT "At L2 - Please wait 3 seconds."
80    WAIT 3
90    GOTO L4
100 L3: PRINT "L3 - GOTO L2"
110   GOTO L2
120 L4: PRINT "L4 - Program stopped."
130   STOP
140 L5: PRINT "Hello, you are at L5."
150   GOTO L1
160   END
```

GRAPIHCS EXAMPLE

```
10  ! CRT Register: 7
20  ! Gets the graphics mode status.
30    CLEAR SCREEN
40    GRAPHICS OFF
50    PRINT "Graphics Mode flag is";STATUS(CRT,7)
60    WAIT 1.5
70    SEPARATE ALPHA FROM GRAPHICS
80    GRAPHICS OFF
90    IF STATUS(CRT,7)=0 THEN
100     GRAPHICS ON
110   END IF
120   MOVE 50,50
130   POLYGON 10,FILL
140   END
```

GRAPIHCS INPUT IS EXAMPLE

```
10    PLOTTER IS CRT,"INTERNAL"
20    GRAPHICS INPUT IS KBD,"KBD"
30    TRACK CRT IS ON
40    FRAME
50    DIGITIZE X,Y,S$
60    PRINT X,Y,S$
70    END
```

GRID EXAMPLE

```
10    CLEAR SCREEN
20    GRID
30    New
40    GRID 20
50    New
60    GRID 20,10
70    New
80    GRID 10,20,20
90    New
100   GRID 20,10,20,10
110   New
120   GRID 10,10,10,10,20
130   New
140   GRID 20,20,10,10,20,20
150   New
160   GRID 10,20,10,20,10,20,10
170   END
180   SUB New
190     WAIT 1.5
200     CLEAR SCREEN
210   SUBEND
```

GSTORE EXAMPLE

```
10    INTEGER A(1:6)
20    GESCAPE CRT,3;A(*)
30    ALLOCATE INTEGER B(1:A(5),1:A(6))
40    GINIT
50    GCLEAR
60    CLEAR SCREEN
70    MOVE 40,40
80    AREA PEN 7
90    RECTANGLE 20,20,FILL
100   GSTORE B(*)
110   WAIT 3
120   CLEAR SCREEN
130   MOVE 30,40
140   GLOAD B(*)
150   END
```

IDN EXAMPLE

```
10    DIM Matrix(1:3,1:3)
20    DATA 1, 2, 3, 4, 5, 6, 7, 8, 9
30    RESTORE
40    READ Matrix(*)
50    CLEAR SCREEN
60    PRINT "The matrix looks like: "
70    Prtmat(Matrix(*),3,3)
80    MAT Matrix=IDN
90    PRINT "Matrix idenity"
100   Prtmat(Matrix(*),3,3)
110   END
120   SUB Prtmat(A(*),Lenarr,Widarr)
130! This sub prints out a matrix length of Lenarr and wide as widarr.
140! A 3x3 matrix would print like:
150!           [ 1 2 3 ]  Widarr = 3
160!           [ 4 5 6 ]
170!           [ 7 8 9 ]
180!           Lenarr = 3
190     ASSIGN @Out TO CRT
200     FOR Col=1 TO Lenarr
210       OUTPUT @Out;"  [";
220       FOR Row=1 TO Widarr
230         OUTPUT @Out;A(Col,Row);
240       NEXT Row
250       OUTPUT @Out;" ]"
260     NEXT Col
270     ASSIGN @Out TO *
280   SUBEND
```

IDRAW EXAMPLE

```
10    CLEAR SCREEN
20    GCLEAR
30    MOVE 0,0
40    DRAW 50,50
50    DRAW 10,50
60    DISP "Program paused."
70    PAUSE
80    CLEAR SCREEN
90    MOVE 0,0
100   IDRAW 50,50
110   IDRAW 10,50
120   END
```

if_then EXAMPLE

```
10    CLEAR SCREEN20    X= NOT 0  ! x is non zero, so it is true.30    PRINT
"Is x true...?"40    IF X THEN50      PRINT "X is true."
60    ELSE
70      PRINT "NO, x is not true."
80    END IF
90    END
```

imag EXAMPLE

```
10    COMPLEX C,Z20    C=CMPLX(3,4)30    Z=CMPLX(3453,4444)40    PRINT
IMAG(Z)50    PRINT REAL(C),IMAG(C)60    END
```

image EXAMPLE

```
10    OPTION BASE 1
20    DIM A(3,3)
30    DATA -4, 36, 2.3, 5, 89, 17,  -6,  -12, 42, 1, 2, 3
40    RESTORE
50  ! Format (Fmt) for specified matrix (3x3)
60 Fmt3x3: IMAGE 3("[",3DD.DD,3DD.DD,3DD.DD,"]",/)
70    CLEAR SCREEN
80    READ A(*)
90    PRINT "Print the array using the 3x3 matrix format/image"
100   PRINT USING Fmt3x3;A(*)
110   END
```

imove EXAMPLE

```
10    CLEAR SCREEN
20    GCLEAR
30    MOVE 0,50
40    PEN 4
50    AREA PEN 1
60    IMOVE 10,10
70    RECTANGLE 5,5,FILL
80    AREA PEN 2
90    IMOVE 30,30
100   RECTANGLE 5,5,FILL
110   PRINT "Press CONTINUE..."
120   PAUSE
130   GINIT
140   CLEAR SCREEN
150   MOVE 0,50
160   PEN 4
170   AREA PEN 1
180   MOVE 10,10
190   RECTANGLE 5,5,FILL
200   AREA PEN 2
210   MOVE 30,30
220   RECTANGLE 5,5,FILL
230   END
```

inmem EXAMPLE

```
10    CLEAR SCREEN
20    Str$="FNZtest"
30    IF INMEM(Str$) THEN
40      PRINT "Procedure ";Str$;" is in memory"
50    ELSE
60      PRINT "No such procedure ";Str$;" in memory."
70    END IF
80    END
90    SUB Test
100   SUBEND
110   DEF FNZtest
120   FNEND
```

inp EXAMPLE

```
10    CLEAR SCREEN
20    PRINT INP(&H3F8)
30    PRINT INP(&H3E8)
40    PRINT INPW(&H3F8)
50    PRINT INPW(&H3E8)
60    OUT (&H3F8),3
70    PRINT INP(&H3F8)
80    OUTW (&H3F8),45
90    PRINT INPW(&H3F8)
100   END
```

input EXAMPLE

```
10    PRINT "Enter your name."
20    INPUT B$
30    PRINT "Enter your age."
40    INPUT A
50    PRINT "Hello ";B$&",";" you are";A;"years old."
60    END
```

inpw EXAMPLE

```
10    PRINT IVAL$(INPW(&H3F80),16)
20    END
```

int EXAMPLE

```
10    J=INT(2.7)
20    PRINT J
30    K=INT(-2.7)
40    PRINT K
50    Number=34.8
60    Gif=INT(Number)
70    PRINT Gif
80    Y=44.54
90    PRINT "greatest integer function=";INT(Y)
100   END
```

integer EXAMPLE

```
10    INTEGER A,B
20    A=10
30    B=5
40    PRINT "A = ";A
50    PRINT "B = ";B
60    END
```

interactive EXAMPLE

```
1      ! The normal functions of the program control keys CLR I/O, ENTER, PAUSE, STEP
2      ! and STOP, are disabled.  The RESET key may also be disabled by specifying the
3      ! optional RESET keyword.  The keys are only disabled while the program is running.
4
10    X=5
20    ON TIME (TIMEDATE+X) MOD 86400 GOTO Here
30    PRINT "I'll wait";X;"seconds. Keys are disabled."
40    SUSPEND INTERACTIVE
50 Loop: GOTO Loop
60 Here: RESUME INTERACTIVE
70    PRINT "Keys re-enabled."
80    END
```

inv EXAMPLE

```
10    DIM Matrix(1:3,1:3)
20    DATA 0, 2, 0, -1, 2, 0, 2, 0, 2
30    RESTORE
40    READ Matrix(*)
50    CLEAR SCREEN
60    PRINT "The matrix looks like: "
70    Prtmat(Matrix(*),3,3)
80    MAT Matrix=INV(Matrix)
90    PRINT "Matrix inverse"
100   Prtmat(Matrix(*),3,3)
110   END
120   SUB Prtmat(A(*),Lenarr,Widarr)
130! This sub prints out a matrix length of Lenarr and wide as widarr.
140! A 3x3 matrix would print like:
150!          [ 1 2 3 ]  Widarr = 3
160!          [ 4 5 6 ]
170!          [ 7 8 9 ]
180!          Lenarr = 3
190     ASSIGN @Out TO CRT
200     FOR Col=1 TO Lenarr
210       OUTPUT @Out;"  [";
220       FOR Row=1 TO Widarr
230         OUTPUT @Out;A(Col,Row);
240       NEXT Row
250       OUTPUT @Out;" ]"
260     NEXT Col
270     ASSIGN @Out TO *
280   SUBEND
```

iplot EXAMPLE

```
10  ! Using IPLOT statements, this program draws an arrow.
20    GINIT
30    IPLOT 50,50
40    IPLOT 10,0
50    IPLOT 0,-10
60    IPLOT -3,3
70    IPLOT -20,-20
80    IPLOT -4,4
90    IPLOT 20,20
100   IPLOT -3,3
110   END
```

ival$ EXAMPLE

```
10    CLEAR SCREEN
20    I=9999
30    PRINT TAB(5),"Convert ";I;"to:"
40    FOR X=1 TO 4
50      DATA 2,8,10,16
60      READ N
70      SELECT N
80      CASE 2
90        PRINT "Binary"
100     CASE 8
110       PRINT "Octal"
120     CASE 10
130       PRINT "Base ten."
140     CASE 16
150       PRINT "Hex."
160     END SELECT
170     PRINT TAB(5),IVAL$(I,N)
180   NEXT X
190   END
```

ival1 EXAMPLE

```
10    CLEAR SCREEN
20    I$="270F"
30    OUTPUT CRT;"converting hex ";I$;" to integer";
40    PRINT IVAL(I$,16)
50    J$="9999"
60    OUTPUT CRT;"converting base 10 ";J$;" to integer";
70    PRINT IVAL(J$,10)
80    K$="23417"
90    OUTPUT CRT;"converting octal ";K$;" to integer";
100   PRINT IVAL(K$,8)
110   L$="0010011100001111"
120   OUTPUT CRT;"converting binary ";L$;" to integer";
130   PRINT IVAL(L$,2)
140   END
```

kbd cmode EXAMPLE

```
10    CLEAR SCREEN
20    KBD CMODE ON  !  Changes softkey compatibility mode to Nimitz
30    Keyten$="KEY 10"
40    SET KEY 10,Keyten$!  Defines softkey 10 as the text:  "KEY 10"
50    END
```

kbd line pen EXAMPLE

```
10    DIM Sometext$[30]
20    CLEAR SCREEN
30    Pencolor=6
40    KBD LINE PEN Pencolor !  Sets the keyboard line pen color
50    PRINT "Input line is in Blue!"!  Output in default ALPHA PEN color
60    INPUT "Input some text",Sometext$        !  Input should appear in Blue
for Pencolor 6
70    PRINT Sometext$
80    PRINT "Output is in default color"!  All output appears in default ALPHA
PEN color
90    END
```

kbd$ EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Type ""A"""
30    ON KBD ALL GOSUB Keyhit           !  defines event branch for keyboard
input
40    REPEAT
50    UNTIL Buf$="A"
60    STOP
70 Keyhit:                              !  Branch taken upon key press
80    Buf$=KBD$                         !  KBD$ returns key to Buf$
90    IF Buf$="A" THEN PRINT "Thank You"
100   RETURN
110   END
```

kbd1 EXAMPLE

```
10    !While the program is running, type a key and its # will be returned to you.
20    CONTROL KBD,203;1
30    CONTROL KBD,204;1
40      PRINT "to quit type q"
50    ON KBD GOSUB Printit
60    LOOP
70       WAIT .01
80    END LOOP
90    Printit:      !
100     K$=KBD
110   IF K$="q" THEN STOP
120   PRINT NUM(K$)       !Prints out the # of the key pressed.
130   RETURN
140   END
```

key labels pen EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Watch Softkey 1"
30    Blue=6                                  !  sets blue
40    White=1                                 !  sets white
50    FOR I=1 TO 100
60      IF I=20 OR I=60 THEN
70        KEY LABELS PEN Blue                 !  sets pen color to blue
80        SET KEY 1,"BLUE"                    !  sets softkey text to "Blue"
90        PRINT "Key Labels are Blue"
100     END IF
110     IF I=40 OR I=80 THEN
120       KEY LABELS PEN White                !  sets pen color to white
130       SET KEY 1,"WHITE"                   !  sets softkey text to "White"
140       PRINT "Key Labels are White"
150     END IF
160     WAIT .025
170   NEXT I
180   SET KEY 1,"EDIT"
190   END
```

key labels EXAMPLE

```
10    CLEAR SCREEN
20    FOR I=1 TO 100
30      IF I=20 OR I=60 THEN
40        KEY LABELS OFF                    !  Turn key labels off
50        PRINT "Soft Key Labels are off"
60        WAIT 1
70      END IF
80      IF I=40 OR I=80 THEN
90        KEY LABELS ON                     !  Turn key labels on
100       PRINT "Soft Key Labels are on"
110       WAIT 1
120     END IF
130     WAIT .025
140   NEXT I
150   END
```

key EXAMPLE

```
10    CLEAR SCREEN
20    LIST KEY
30    END
```

knob EXAMPLE

```
10    ON KNOB 1 GOSUB Here
20    CLEAR SCREEN
30    FOR Loop=1 TO 10
40      WAIT 1          !Move the mouse to run program properly.
50    NEXT Loop
60    STOP
70 Here: PRINT KNOBX,KNOBY  !Print out the amount moved in the x and y
directions.
80    RETURN
90    END
```

knobx EXAMPLE

```
10    ON KNOB 1 GOSUB Here
20    CLEAR SCREEN
30    FOR Loop=1 TO 10
40      WAIT 1        !Move the mouse when the program is run.
50    NEXT Loop
60    STOP
70 Here: PRINT KNOBX,KNOBY  !Indicates the amount the mouse moved in the x-y
plane.
80    RETURN
90    END
```

knoby EXAMPLE

```
10    ON KNOB 1 GOSUB Here
20    CLEAR SCREEN
30    FOR Loop=1 TO 10
40      WAIT 1        !Move the mouse when the program is run.
50    NEXT Loop
60    STOP
70 Here: PRINT KNOBX,KNOBY  !Indicates the amount the mouse moved in the x-y
plane.
80    RETURN
90    END
```

label EXAMPLE

```
10    CLEAR SCREEN
20    FOR Height=1 TO 12
30      MOVE 40,40
40      DISP Height
50      CSIZE Height    !Change the size of the label.
60      LABEL "Hello World!"
70      WAIT 1
80      CLEAR SCREEN
90    NEXT Height
100   END
```

ldir EXAMPLE

```
10    GINIT
20    MOVE 40,40
21    Mode=1
30 !LABEL "Hello" prints huge
40    IF Mode THEN
50      DEG ! defaults to radians
60      PEN 7
70      Circ
80    END IF
90    END
100  SUB Circ
110    FOR X=0 TO 360 STEP 10
120      LDIR X
130      LABEL "Hello"  ! small print
140    NEXT X
150  SUBEND
```

len EXAMPLE

```
10    DIM Name$[20]
20    CLEAR SCREEN
30    Name$="Hello"
40    Len_name=LEN(Name$)
50    PRINT Name$;" is";Len_name;"chars long - using LEN"
60    END
```

let EXAMPLE

```
10    LET X=8
20    PRINT X
30    END
```

lexical order is EXAMPLE

```
10    LEXICAL ORDER IS ASCII
20    PRINT SYSTEM$("LEXICAL ORDER IS")
30    END
```

lgt EXAMPLE

```
10    X=10000
20    COMPLEX C
30    C=CMPLX(3,5)
40    CLEAR SCREEN
50    PRINT "The Log (base 10) of";X;"is";LGT(X)
60    END
```

line type EXAMPLE

```
10    GINIT
20    PEN 6
30    FOR Loop=1 TO 10
40      MOVE 40+(Loop*10),40
50      DRAW 40+(Loop*10),80
60      DISP "Line type: ";Loop
70      PAUSE
80      LINE TYPE Loop
90    NEXT Loop
100   END
```

linput EXAMPLE

```
10    DIM Array$(3)[100]
20    LINPUT "Please enter a name",Array$(I)
30    PRINT Array$(*)
40    END
```

list bin EXAMPLE

```
10    CLEAR SCREEN
20    LIST BIN
30    END
```

list key EXAMPLE

```
10    CLEAR SCREEN
20    LIST KEY
30    END
```

list EXAMPLE

```
10    CLEAR SCREEN
20    LIST 20,30
30    END
```

load bin EXAMPLE

```
10    LOAD BIN "SERIAL"
20    LIST BIN
30    END
```

load key EXAMPLE

```
10    CLEAR SCREEN
20    STORE KEY "keys2"
30    READ KEY 1,Str$
40    PRINT "Key 1 was ";Str$;" now is QUIT -- Please type CONT"
50    SET KEY 1,"QUIT"
60    PAUSE
70    LOAD KEY "keys2"
80    PRINT "Keys are returned to normal."
90    PURGE "keys2"
100   END
```

load EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "CONT to LOAD file."
30    PAUSE
40    LOAD "print.prg"
50    END
```

loadsub EXAMPLE

```
10    LOADSUB "New" FROM "grid.prg"
20    PAUSE
30    PRINT "Paused. Press CONT"
40    New
50    PRINT "All finished."
60    END
```

locator EXAMPLE

```
10    CLEAR SCREEN
20    SET LOCATOR 0,0
30    WHILE KBD$=""
40      READ LOCATOR X,Y,Stat$
50      PRINT X,Y
60      PRINT "Status: ";Stat$;" - length: ";LEN(Stat$)
70      WAIT 1
80    END WHILE
90    END
```

lock EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** Lock Test ****"
30    PRINT
40    CREATE "test.txt",100
50    DIM Test$[100]
60    ASSIGN @File TO "test.txt";FORMAT ON
70    PRINT "CAT after assignment..."
80    CAT "test.txt";NO HEADER
90    OUTPUT @File;"This is the contents of test.txt"
100   RESET @File
110   ENTER @File;Test$
120   PRINT
130   PRINT Test$
140   PRINT
150   LOCK @File;CONDITIONAL Error                         !This locks a file
while you are working on it.
160   PRINT "CAT after lock..."
170   CAT "test.txt";NO HEADER
180   PRINT
190   PRINT "Lock result is";Error
200   UNLOCK @File                                         !This unlocks the
file.
210   PRINT
220   PRINT "CAT after unlock..."
230   CAT "test.txt";NO HEADER
240   ASSIGN @File TO *                                    !This will also
unlock the file. Comment out the unlock
250   PRINT                                                !statement to show
this.
260   PRINT "CAT after file is closed..."
270   CAT "test.txt";NO HEADER
280   PURGE "test.txt"
290   END
```

log EXAMPLE

```
10    CLEAR SCREEN
20    IF LOG(EXP(65))<>65 THEN PRINT "Test failed."
30    RAD     ! Complex calculations are always done in radians!
40    COMPLEX C
50    C=CMPLX(4,7)
60    X=REAL(LOG(C))
70    Y=LOG(ABS(C))
80    PRINT X;"=";Y;"?"
90    IF X=Y THEN
100     PRINT "True"
110   ELSE
120     PRINT "False - error in documentation."
130   END IF
140   X=IMAG(LOG(C))
150   Y=ARG(C)
160   PRINT X;"=";Y;"?"
170   IF X=Y THEN
180     PRINT "True"
190   ELSE
200     PRINT "False - error in documentation."
210   END IF
220   END
```

loop EXAMPLE

```
10    CLEAR SCREEN
20    LOOP
30      PRINT "Iterator: ";Counter
40      PRINT Counter MOD 3
50      Counter=Counter+1
60    EXIT IF Counter=5 OR (Counter MOD 4)=3
70      PRINT "Not finished."
80    END LOOP
90    PRINT "All done."
100  END
```

lorg EXAMPLE

```
10    GINIT
20    CLEAR SCREEN
30    FOR X=1 TO 9
40      MOVE 70,40
50      LORG X
60      LABEL RPT$(" ",5)&VAL$(X)&"Hi"&RPT$(" ",5)
70    NEXT X
80    END
```

lwc$ EXAMPLE

```
10    CLEAR SCREEN
20    Test$="HELLO"
30    PRINT Test$;" is in all caps."
40    Test$=LWC$(Test$)
50    PRINT "LCW$ turns them all lower case: ";Test$
60    END
```

mass storage is EXAMPLE

```
10  ! Comments: MASS STORAGE IS, MSI, CD  work correctly. Both MSI
20  ! and CD convert to MASS STORAGE IS.
30    CLEAR SCREEN
40    PRINT "*** CD or MSI TEST ****"
50    PRINT "MSI: ",SYSTEM$("MSI")
60    MASS STORAGE IS "C:\"
70    PRINT "MSI: ",SYSTEM$("MSI")
80    MASS STORAGE IS "d:\"
90    PRINT "MSI: ",SYSTEM$("MSI")
100   END
```

mat reorder EXAMPLE

```
10    OPTION BASE 1
20    DIM Matrix(3,3),Vector(3)
30    DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 3, 2, 1
40    RESTORE
50    READ Matrix(*),Vector(*)
60    CLEAR SCREEN
70    PRINT "The matrix looks like: "
80    Prtmat(Matrix(*),3,3)
90    MAT REORDER Matrix BY Vector,2
100   PRINT "MAT reorder"
110   Prtmat(Matrix(*),3,3)
120   END
130   SUB Prtmat(A(*),Lenarr,Widarr)
140! This sub prints out a matrix length of Lenarr and wide as widarr.
150! A 3x3 matrix would print like:
160!           [ 1 2 3 ]  Widarr = 3
170!           [ 4 5 6 ]
180!           [ 7 8 9 ]
190!           Lenarr = 3
200     ASSIGN @Out TO CRT
210     FOR Col=1 TO Lenarr
220       OUTPUT @Out;"  [";
230       FOR Row=1 TO Widarr
240         OUTPUT @Out;A(Col,Row);
250       NEXT Row
260       OUTPUT @Out;" ]"
270     NEXT Col
280     ASSIGN @Out TO *
290   SUBEND
```

mat search EXAMPLE

```
10     CLEAR SCREEN
20     OPTION BASE 1
30     DIM Numbers(11)
40     DATA 6, 1, 9, 2, 8, 3, 8, 9, 1, 7, 5
50     RESTORE
60     READ Numbers(*)
70     PRINT "The numbers read:"
80     PRINT Numbers(*)
90     PRINT
100
110    MAT SEARCH Numbers,MAX;Max
120    MAT SEARCH Numbers,LOC MAX;Loc_max
130    MAT SEARCH Numbers,MIN;Min
140    MAT SEARCH Numbers,LOC MIN;Loc_min
150    MAT SEARCH Numbers,# LOC (Max);Num_max
160    MAT SEARCH Numbers,# LOC (Min);Num_min
170    MAT SEARCH Numbers,LOC (<2);Loc_num,4
180
190    PRINT "Maximum value: ";Max
200    PRINT "It first occurs in element: ";Loc_max
210    PRINT "It occurs ";Num_max;" times."
220    PRINT "Minimum value: ";Min
230    PRINT "It is found in element: ";Loc_min
240    PRINT "And occurs ";Num_min;" times."
250    PRINT "First occurence of a number < 2 starting from element 4"
260    PRINT "is in array element: ";Loc_num
270    END
```

mat sort EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** TEST ****"
30    DIM A(1:4),B(3)
40    DATA 5, 9, 2, 8, 6, 9, 0, 1
50    RESTORE
60    READ A(*)
70    PRINT "A = ";A(*)
80    MAT SORT A(*)
90    PRINT "Array A sorted"
100   PRINT "A = ";A(*)
110   MAT SORT A(*) DES
120   PRINT "Array A sorted in descending order."
130   PRINT "A = ";A(*)
140   PRINT
150   READ B(*)
160   PRINT "B = ";B(*)
170   MAT SORT B(*) TO B
180   PRINT "Sorting B to A gives:"
190   PRINT "B = ";B(*)
200   END
```

mat EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** TEST ****"
30    DIM A(1:4)
40    DATA 5, 9, 2, 8
50    RESTORE
60    READ A(*)
70    PRINT "A = ";A(*)
80    MAT SORT A(*)
90    PRINT "Array A sorted"
100   PRINT "A = ";A(*)
110   MAT SORT A(*) DES
120   PRINT "Array A sorted in descending order."
130   PRINT "A = ";A(*)
140   END
```

max EXAMPLE

```
10  ! This is your basic array usage.
20    DIM A(1:4)
30    DATA 5, 6, 7, 3
40    RESTORE
50    READ A(*)
60    PRINT "The maximum value in the array is";MAX(A(*))
70    END
```

maxlen EXAMPLE

```
10    DIM Name$[20]
20    CLEAR SCREEN
30    Name$="Hello"
40    Len_name=MAXLEN(Name$)  !Returns the max length the string can be
according to DIM.
50    PRINT Name$;" is";Len_name;"chars long - using MAXLEN"
60    Len_name=LEN(Name$)
70    PRINT Name$;" is";Len_name;"chars long - using LEN"
80    END
```

maxreal EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "The largest positive real number is";MAXREAL
30    END
```

merge alpha EXAMPLE

```
10    MERGE ALPHA WITH GRAPHICS
20    CLEAR SCREEN
30    PRINT "With Alpha and Graphics merged, the text and"
40    PRINT "graphics should disappear with a CLEAR SCREEN"
50    PAUSE
60    MOVE 0,60
70    AREA PEN 4
80    RECTANGLE 30,30,FILL
90    WAIT 2
100   SEPARATE ALPHA FROM GRAPHICS
110   CLEAR SCREEN
120   PRINT "This text should disappear with a CLEAR SCREEN"
130   PRINT "The square should not be erased"
140   MOVE 0,60
150   AREA PEN 7
160   RECTANGLE 30,30,FILL
170   WAIT 2
180   CLEAR SCREEN
190   WAIT 1
200   PRINT "Notice how the square was not erased!"
210   END
```

min EXAMPLE

```
10  ! This is your basic array usage.
20    DIM A(1:4)
30    DATA 5, 6, 7, 3
40    RESTORE
50    READ A(*)
60    PRINT "The minimum value in the array is";MIN(A(*))
70    END
```

minreal EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "The smallest positive real number is";MINREAL
30    END
```

mod EXAMPLE

```
10    CLEAR SCREEN
20    X=5
30    Y=5
40    PRINT "The";X;"MOD";Y;"is";X MOD Y
50    PRINT "The";X;"MODULO";Y;"is";X MODULO Y
60    END
```

modulo EXAMPLE

```
10    CLEAR SCREEN
20    INTEGER X,Y
30    X=5
40    Y=-85
41    PRINT X-Y*(INT(X/Y))
50    PRINT "The";X;"MOD ";Y;"is ";X MOD Y
60    PRINT "The";X;"MODULO ";Y;"is ";X MODULO Y
70    END
```

move EXAMPLE

```
10    CLEAR SCREEN
20    GINIT
30    MOVE 20,20
40    RECTANGLE 10,10
50    MOVE 50,50
60    POLYGON 10
70    PRINT "test complete."
80    END
```

msi EXAMPLE

```
10  ! Comments: MASS STORAGE IS, MSI, CD  work correctly. Both MSI
20  ! and CD convert to MASS STORAGE IS.
30    CLEAR SCREEN
40    PRINT "*** CD or MSI TEST ****"
50    PRINT "MSI: ",SYSTEM$("MSI")
60    MASS STORAGE IS "C:\"
70    PRINT "MSI: ",SYSTEM$("MSI")
80    MASS STORAGE IS "D:\"
90    PRINT "MSI: ",SYSTEM$("MSI")
100   END
```

next EXAMPLE

```
10    FOR J=40 TO 500 STEP 20
20      PRINT J
30    NEXT J
40    END
```

not EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Not 1 is";NOT 1
30    PRINT "Not 0 is";NOT 0
40    END
```

npar EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** Optional parameter TEST ****"
30    Bigparams(1,2)
40    STOP        !NPAR counts the # of parameters sent to a SUB.
50    END
60    SUB Bigparams(A,B, OPTIONAL C,D)
70      PRINT NPAR;"parameters sent to SUB"
80    SUBEND
```

num1 EXAMPLE

```
10    CLEAR SCREEN
20    Str$="Hello"
30    PRINT "NUM returns the ASCII value of the "
40    PRINT "first character in a string."
50    PRINT
60    PRINT "For example, the string: ";Str$
70    X=NUM(Str$)
80    PRINT "The value returned was: ";X;"or: ";CHR$(X)
90    END
```

off cycle1 EXAMPLE

```
10    ON CYCLE 5 GOTO Here  !Wait 5 seconds then go to line 160.
20    Start=TIMEDATE
30    CLEAR SCREEN
40    LOOP
50      PRINT "Waiting"
60      WAIT 1
70      IF TIMEDATE-5>Start THEN !If it waits longer than 5 seconds, then test
will fail.
80        PRINT "Cycle is off."
90        OFF CYCLE
100     END IF
110     IF TIMEDATE-6>Start THEN
120       PRINT "Forcing stop."
130       STOP
140     END IF
150   END LOOP
160 Here: PRINT "OFF CYCLE worked correctly."
170   END
```

off delay EXAMPLE

```
10    CLEAR SCREEN
20    ON DELAY 3 GOTO Here !Wait 3 seconds and then go to line 100.
30    PRINT "ON Delay"
40    WAIT 2
50    OFF DELAY            !Turn off the branch event to line 100.
60    PRINT "OFF Delay"
70    WAIT 2
80    PRINT "Test passed."
90    STOP
100 Here: PRINT "OFF DELAY did not work properly."
110   END
```

off end EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** output TEST ****"
30    CREATE "test.txt",0
40    ASSIGN @File TO "test.txt";FORMAT ON
50    ON END @File GOTO Here
60    OUTPUT @File;"This is a test"
70    OUTPUT @File;"This is a test - line 2"
80    RESET @File
90    FOR Loop=1 TO 5
100     ENTER @File;Test$
110     PRINT Test$,Loop
120     ON ERROR GOTO 190
130     IF Loop=1 THEN OFF END @File !This causes an error after first loop iteration.
140   NEXT Loop
150 Here: !
160   PRINT "OFF END command did not work properly"
170   ASSIGN @File TO *
180   STOP
190   OFF ERROR
200   ASSIGN @File TO *
210   PRINT "Test Passed"
220   PURGE "test.txt"
230   END
```

off error EXAMPLE

```
10    !This program should cause an error.
20    ON ERROR GOTO Here
30    PRINT "I want to get an error."
40    CAUSE ERROR 80
50    STOP
60 Here: PRINT "Program should reach here when it errors."
70    CLEAR ERROR
80    END
```

off intr EXAMPLE

```
10    RESET 7
30 Topp: ENABLE INTR 7;2     ! RESPONDS TO SRQ
40    ON INTR 7,1 GOTO Intrr
50    ON TIMEOUT 7,30 GOTO Stopp
60    LOOP
70      OUTPUT 720;"HELLO"
80    END LOOP
90    STOP
100 Stopp:!
110   PRINT "TIMED OUT"
120   STOP
130 Intrr:!
140   PRINT "INTERRUPTED"
150   OFF INTR
160   GOTO Topp
170   END
```

off kbd EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Type ""A"""
30    ON KBD ALL GOSUB Keyhit           !  defines event branch for keyboard
input
40    REPEAT
50      IF Buf$="x" THEN                !  Type x to use OFF KBD.
60        OFF KBD
70        Disabled=500
80        PRINT "OFF KBD."
90        Buf$="A"
100     ELSE
110       DISP Buf$
120     END IF
130   UNTIL Buf$="A"
140   STOP
150 Keyhit:                             !  Branch taken upon key press
160   Buf$=KBD$                         !  KBD$ returns key to Buf$
170   IF Buf$="A" THEN PRINT "Thank You"
180   RETURN
190   END
```

off key EXAMPLE

```
10    ON KEY 7 GOTO Here
20    ON DELAY 10 GOTO Stophere
30    PRINT "Press F7 ONLY three times in 10 seconds."
40    LOOP
50 Loop: GOTO Loop
60 Here: PRINT "F7 pressed"
70      Counter=Counter+1
80      IF Counter=3 THEN OFF KEY
90    END LOOP
100 Stophere: PRINT "OFF KEY works properly."
110   END
```

off knob EXAMPLE

```
10    ON KNOB 1 GOSUB Here
20    DISP "Move the mouse"
30    CLEAR SCREEN
40    FOR Loop=1 TO 10
50      WAIT 1
60      IF Loop=5 THEN
70        OFF KNOB
80        PRINT "OFF KNOB - no more mouse movements accepted."
90      END IF
100     PRINT Loop
110   NEXT Loop
120   STOP
130 Here: PRINT KNOBX,KNOBY
140   RETURN
150   END
```

off signal EXAMPLE

```
10    ON SIGNAL 5 GOTO Here
20 Loop:!
30    FOR X=1 TO 10
40      PRINT X
50      IF X=9 THEN SIGNAL 5
60      IF X=5 THEN OFF SIGNAL
70    NEXT X
80 Here: IF X=11 THEN
90      PRINT "OFF SIGNAL worked."
100   ELSE
110     PRINT X
120     PRINT "OFF SIGNAL did not function properly."
130   END IF
140   PRINT "Test complete"
150   END
```

off time EXAMPLE

```
10    X=5
20    ON TIME (TIMEDATE+X) MOD 86400 GOTO Here !After 5 seconds, go to line
80.
30    PRINT "I'll wait";X;"seconds."
40    WAIT 2
50    OFF TIME
60    PRINT "OFF TIME worked properly."
70    STOP
80 Here: PRINT "Should never reach this point."
90    END
```

off timeout EXAMPLE

```
10    ! LOAD BIN "SERIAL32"
20    ON TIMEOUT 9,5 GOTO L50
30    ON DELAY 3 GOTO L22
40 L20: PRINT "WAITING..."
50 L21: ENTER 9;X$
60 L22: OFF TIMEOUT
70    PRINT "Off time out."
80    STOP
90    PRINT X$
100   GOTO L20
110 L50: PRINT "IT TIMED OUT"
120   END
```

off EXAMPLE

```
10 ! CRT Register: 7
20 !    Get the graphics mode status.
30    CLEAR SCREEN
40    GRAPHICS OFF
50    PRINT "Graphics Mode flag is";STATUS(CRT,7)
60    END   !Should print out 0.
```

on cycle EXAMPLE

```
10    ON CYCLE 5 GOTO Here  !After 5 seconds, go to line 70.
20    CLEAR SCREEN
30    LOOP
40      PRINT "Still waiting"
50      WAIT 1
60    END LOOP
70 Here: PRINT "ON CYCLE worked."
80    END
```

on delay EXAMPLE

```
10    CLEAR SCREEN
20    ON DELAY 3 GOTO Here  ! Wait 3 seconds.
30    PRINT "I shall wait 3 seconds."
40 Loop: GOTO Loop
60 Here: PRINT "ON DELAY worked."
70    END
```

on end EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** output TEST ****"
30    CREATE "test.txt",0
40    ASSIGN @File TO "test.txt";FORMAT ON
50    ON END @File GOTO Here
60    OUTPUT @File;"This is a test","Second line"
70    RESET @File
80    FOR Loop=1 TO 5
90      ENTER @File;Test$
100     PRINT Test$,Loop
110   NEXT Loop
120 Here: !
130   PRINT "End of file reached."
140   ASSIGN @File TO *
150   PURGE "test.txt"
160   END
```

on eot EXAMPLE

```
10    DIM Buff$[60] BUFFER  !for this program, you need to be hooked up 
serialy to another
20    DIM A$[60]            !computer with one running this program, and the 
other sending
30    CLEAR SCREEN          !data to this program through an output statement.
40    RESET 9
50    PRINT "*** End Of Transfer Test ***"
60    ASSIGN @Buf TO BUFFER Buff$
70    ASSIGN @In TO 9
80    ON EOT @In GOTO Alldone
90    TRANSFER @In TO @Buf
100   ENTER Buff$;A$
110   PRINT A$
120 Loop: GOTO Loop
130 Alldone: PRINT "The transfer is completed."
140   PRINT "This is the contents of the buffer"
150   PRINT Buff$
160   END
```

on error EXAMPLE

```
10    ON ERROR GOTO Here
20    CAUSE ERROR 0
30    PRINT "ON ERROR did not work."
40    STOP
50 Here: PRINT "ON ERROR functioned properly."
60    END
```

on intr EXAMPLE

```
10    CLEAR SCREEN
30    PRINT "Press the SRQ"
40    RESET 7
50    ENABLE INTR 7;2   ! RESPONDS TO SRQ
60    ON INTR 7,1 GOTO Intrr
70    ON DELAY 30 GOTO Stopp
80    LOOP
90      OUTPUT 720;"HELLO"
100   END LOOP
110   STOP
120 Stopp:!
130   PRINT "TIMED OUT"
140   STOP
150 Intrr:!
160   PRINT "INTERRUPTED"
170   END
```

on kbd1 EXAMPLE

```
10    CONTROL KBD,203;1
20    CONTROL KBD,204;1
30    ON KBD ALL GOSUB Keyhit   !  defines event branch for keyboard input
40    CLEAR SCREEN
50    PRINT "Type A"
60    REPEAT
70    UNTIL Buf$="A"
80    STOP
90 Keyhit:                      !  Branch taken upon key press
100   Buf$=KBD$                 !  KBD$ returns key to Buf$
110   IF Buf$="A" THEN
120       PRINT "Thank You"
130       DISP "Test complete."
140       RETURN
150   ELSE
160       BEEP
170       DISP Buf$
180       RETURN
190   END IF
200   END
```

on key EXAMPLE

```
10    ON KEY 7 GOTO Here
20    LOOP
30      PRINT "Press F7"
40 Loop: GOTO Loop
50 Here: PRINT "ON KEY worked properly."
60      Counter=Counter+1
70    EXIT IF Counter=1
80    END LOOP
90    END
```

on knob EXAMPLE

```
10    ON KNOB 1 GOSUB Here
20    CLEAR SCREEN
30    FOR Loop=1 TO 10
40      WAIT 1        !Move the mouse when the program is run.
50    NEXT Loop
60    STOP
70 Here: PRINT KNOBX,KNOBY  !Indicates the amount the mouse moved in the x-y
plane.
80    RETURN
90    END
```

on signal EXAMPLE

```
10    ON SIGNAL 5 GOTO Here
20 Loop:!
30    FOR X=1 TO 10
40      PRINT X
50      IF X=5 THEN SIGNAL 5
60    NEXT X
70 Here: IF X=5 THEN PRINT "ON SIGNAL worked."
80    PRINT "Test complete"
90    END
```

on time EXAMPLE

```
10    X=5
20    ON TIME (TIMEDATE+X) MOD 86400 GOTO Here
30    PRINT "I'll wait";X;"seconds."
40 Loop: GOTO Loop
50 Here: PRINT "ON TIME worked properly."
60    END
```

on timeout EXAMPLE

```
10    ! LOAD BIN "SERIAL32"
20    ON TIMEOUT 9,5 GOTO L50  ! Wait 5 seconds and then timeout.
30 L20: PRINT "WAITING..."
40    ENTER 9;X$
50    PRINT X$
60    GOTO L20
70 L50: PRINT "IT TIMED OUT"
80    END
```

on EXAMPLE

```
10    X=2
20    ON X GOTO L1,L2
30 L1: PRINT "Line one."
40 L2: PRINT "Line two."
50    END
```

option base EXAMPLE

```
10    DATA 0, 1, 2, 3, 4, 5
20    CLEAR SCREEN
30    OPTION BASE 0
40    PRINT "Option base is 0"
50    REAL A(5)
60    PRINT "Declared array 5 elements."
70    READ A(*)
80    PRINT A(*)
90    PRINT "Printed 6 items."
100   PRINT
110   New
120   END
130   SUB New
140     DATA 1, 2, 3
150     OPTION BASE 1
160     PRINT "Option base is 1"
170     REAL A(3)
180     PRINT "Declared array of 3."
190     READ A(*)
200     PRINT A(*)
210   SUBEND
```

optional EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "5 + 8 =";FNAdd(5,8)
30    PRINT FNMessage$
40    PRINT FNMessage$("Hello")
50    END
60    DEF FNAdd(A,B)
70      RETURN A+B
80    FNEND
90    DEF FNMessage$(OPTIONAL String$)
100     IF NPAR=0 THEN RETURN "OPTIONAL parameter not used."
110     RETURN "OPTIONAL parameter used."
120   FNEND
```

or EXAMPLE

```
10  ! This program prints the truth table for an OR funtion.
20    DATA 0,0,0,1,1,0,1,1
30    RESTORE
40    CLEAR SCREEN
50    PRINT TAB(10),"OR test"
60    PRINT " J"," K","J OR K"
70    FOR L=1 TO 4
80      READ J,K
90      PRINT J,K,J OR K
100   NEXT L
110   END
```

out EXAMPLE

```
10    OUT &H300,64+16
20    END
```

outp EXAMPLE

```
10    CLEAR SCREEN
20    PRINT INP(&H3F8)
30    PRINT INP(&H3E8)
40    PRINT INPW(&H3F8)
50    PRINT INPW(&H3E8)
60    OUT (&H3F8),3
70    PRINT INP(&H3F8)
80    OUTW (&H3F8),45
90    PRINT INPW(&H3F8)
100   END
```

output EXAMPLE

```
10    DIM R(1),A$(1)[1]
20    R(0)=-1
30    R(1)=+1
40    MAT A$=("A")
50    ASSIGN @I TO CRT
60    OUTPUT @I;1.E+5,1.E+7
70    OUTPUT @I;1;-1
80    OUTPUT @I;R(*),
90    OUTPUT @I;CMPLX(1,1.23456789012345E+7)
100   OUTPUT @I;CMPLX(1,1);
110   OUTPUT @I;"B";"C","D"
120   OUTPUT @I;A$(*);
130   END
```

outw EXAMPLE

```
10    OUTW Base+3,&HF001
20    END
```

pass control EXAMPLE

```
10    PASS CONTROL 720
20    END
```

pause EXAMPLE

```
10    PRINT "Pausing. Press CONT to continue..."
20    PAUSE
30    PRINT "I'm done"
40    END
```

pdir EXAMPLE

```
10    CLEAR SCREEN
20    GINIT
30    MOVE 40,40
40    PDIR PI/4  !radians
50    RECTANGLE 10,20
60    DISP "1"
70    WAIT 1
80    AREA PEN 3
90    RECTANGLE 10,-20,FILL
100   DISP "2"
110   WAIT 1
120   PEN 2
130   RECTANGLE -10,-20,EDGE
140   DISP "3"
150   WAIT 1
160   AREA PEN 7
170   PEN 8
180   RECTANGLE -10,20,FILL,EDGE
190   DISP "4"
200   WAIT 1
210   DISP
220   END
```

pen EXAMPLE

```
10    GINIT
20    GCLEAR
30    PLOTTER IS CRT,"INTERNAL";COLOR MAP
40    MOVE 40,40
50    AREA PEN 6
60    FOR L=-5 TO 5
70      DISP L
80      PEN L
90      RECTANGLE 90,30,FILL,EDGE
100     WAIT 1
110   NEXT L
120   END
```

penup EXAMPLE

```
10    CLEAR SCREEN
20    PLOT 40,40,-1
30    PENUP
40    PLOT 80,80,2
50    PLOT 80,90
60    END
```

pi EXAMPLE

```
10    PRINT "The value of pi is close to ";PI
20    END
```

pivot EXAMPLE

```
10    CLEAR SCREEN
20    GINIT
30    MOVE 40,40
40    DEG
50    PIVOT 45
60    RECTANGLE 10,20
70    DISP "1"
80    WAIT 1
90    AREA PEN 3
100   RECTANGLE 10,-20,FILL
110   DISP "2"
120   WAIT 1
130   PEN 2
140   RECTANGLE -10,-20,EDGE
150   DISP "3"
160   WAIT 1
170   AREA PEN 7
180   PEN 8
190   RECTANGLE -10,20,FILL,EDGE
200   DISP "4"
210   WAIT 1
220   DISP
230   END
```

plot EXAMPLE

```
10    CLEAR SCREEN
20    PLOT 40,10,-1 ! lower pen before move
30    PENUP
40    PLOT 80,80    ! lower after move, default is 1
50    PLOT 80,90,0  ! raise after move
60    PLOT 90,80! pen is down
70    PLOT 100,60,-1  ! lower pen before move
80    PLOT 85,40! pen is down
90    PLOT 0,0,-2  ! raise before move
100   END
```

plotter is EXAMPLE

```
10    PLOTTER IS CRT,"INTERNAL";COLOR MAP
20    AREA PEN 6
30    RECTANGLE 30,40,FILL,EDGE
40    END
```

polygon EXAMPLE

```
10    CLEAR SCREEN
20    MOVE 40,40
30    POLYGON 5
40    POLYGON 10,6
50    POLYGON 15,12,5
60    MOVE 80,40
70    POLYGON 20,FILL
80    MOVE 30,80
90    POLYGON 15,FILL,EDGE
100   END
```

polyline EXAMPLE

```
10    CLEAR SCREEN
20    MOVE 40,40
30    POLYLINE 5
40    POLYLINE 10,6
50    POLYLINE 15,12,5
60    END
```

pos EXAMPLE

```
10    CLEAR SCREEN
20    Name$="Hello"
30    PRINT "Using the string ";Name$
40    PRINT "The position where 'll' is located is"
50    PRINT POS(Name$,"ll")
60    END
```

print pen EXAMPLE

```
10    CLEAR SCREEN
20    PLOTTER IS CRT,"INTERNAL";COLOR MAP
30    PRINT "*** TEST ****"
40    FOR Numloop=0 TO 25
50      PRINT PEN Numloop
60      PRINT "*";Numloop
70    NEXT Numloop
80    END
```

print EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** Print TEST ****"
30    PRINT "[TAB]";TAB(15);"15 spaces"
40    PRINT TABXY(5,5);"TABXY test"
50    END
```

printall EXAMPLE

```
10    CREATE "test.txt",0
20    PRINTALL IS "test.txt"
30    PRINT "Hello, this is a test."
40    PRINT SYSTEM$("I am Bill Gates")
50    PAUSE
60    PURGE "test.txt"
70    PRINT "File purged."
80    END
```

printer is EXAMPLE

```
10    CLEAR SCREEN
20    PRINT PRT
30    PRINTER IS PRT
40    PRINT "Hello"
50    PRINTER IS CRT
60    PRINT "Hello"
70    END
```

priority EXAMPLE

```
10    CLEAR SCREEN
20    FOR Loop=0 TO 15
30      SYSTEM PRIORITY Loop
40      PRINT "System Priority: "&SYSTEM$("SYSTEM PRIORITY")
50    NEXT Loop
60    Testme
70    PRINT "Back in main."
80    PRINT "Priority: "&SYSTEM$("SYSTEM PRIORITY")
90    END
100   SUB Testme
110     PRINT "In test SUB; setting priority to 1"
120     SYSTEM PRIORITY 1
130     PRINT "Priority: "&SYSTEM$("SYSTEM PRIORITY")
140   SUBEND
```

protect EXAMPLE

```
10    CLEAR SCREEN
20    MASS STORAGE IS "C:\"
30    PRINT "Creating file"
40    SAVE "junk.XXX"
50    WAIT 2
60    CLEAR SCREEN
70    PROTECT "junk.XXX","R"  !Protect the file as a read-only file.
80    PRINT "Making junk.XXX a read-only file."
90    WAIT 3
100   CLEAR SCREEN
110   PRINT "Unprotecting junk.XXX"
120   PROTECT "junk.XXX",""
130   WAIT 2
140   PURGE "junk.XXX"
150   PRINT "File purged"
160   END
```

pround EXAMPLE

```
10    Number=656576.2346516
20    CLEAR SCREEN
30    PRINT "Rounding the number:";Number
40    FOR Roundto=-6 TO 6
50      PRINT Roundto,PROUND(Number,Roundto)
60    NEXT Roundto
70    END
```

prt EXAMPLE

```
10    CLEAR SCREEN
20    PRINT PRT
30    PRINTER IS PRT
40    PRINT "Hello"
50    END
```

purge EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Saving code. Creating file"
30    SAVE "file.txt"
40    CAT "*.txt";NAMES
50    PRINT "Press CONT to purge file."
60    PAUSE
70    PRINT "Now, I will purge it."
80    PURGE "file.txt"
90    CAT "*.txt";NAMES
100   PRINT "File purged."
110   END
```

quit EXAMPLE

```
10    PRINT "If this works right after you CONT, the HTBasic child window will
close."
20    PRINT "Program paused..."
30    PAUSE
40    QUIT
50    END
```

quit all EXAMPLE

```
10    PRINT "If this works right after you CONT,  HTBasic will quit."
20    PRINT "Program paused..."
30    PAUSE
40    QUIT ALLr
50    END
```

rad EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Testing the trigonometry mode."
30    OUTPUT CRT;"We are currently in";
40    OUTPUT CRT;" radian ";
50    PRINT "mode."
60    PRINT
70    PRINT "Please enter the desired input as mentioned below."
80    INPUT "Radians to convert to degrees",A
90    A=A*180/PI
100   PRINT "That is ";A;"degrees."
110   END
```

randomize EXAMPLE

```
10    Count=0
20    RANDOMIZE
30    CLEAR SCREEN
40    REPEAT
50      WAIT .5
60      Number=INT(RND*100)
70      PRINT Number
80      Count=Count+1
90    UNTIL Count=10
100   PRINT "Random number test done."
110   END
```

rank EXAMPLE

```
10    OPTION BASE 1
20    DIM A(16,6)
30    DIM B(5,7,3)
40    CLEAR SCREEN
50    Pass_a(A(*))
60    Pass_a(B(*))
70    END
80    SUB Pass_a(REAL A(*))
90      L=RANK(A)
100     PRINT "The array passed in has the following rank."
110     PRINT "Rank: ";L
120   SUBEND
```

ratio EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "X width:",CHRX,"y height:",CHRY,"Ratio:",RATIO
30    END
```

read key EXAMPLE

```
10    ! Comments: Uncomment the SET KEY option if no
20    ! softkeys have been defined. WARNING: Do not
30    ! uncomment if you do not want your softkeys
40    ! changed!!!
50    DIM A$[30]
60    CLEAR SCREEN
70    PRINT "*** TEST ****"
80    FOR Nloop=1 TO 22
90         !SET KEY Nloop,"CLS"
100     READ KEY Nloop,A$
110     PRINT Nloop,A$
120     WAIT 1
130   NEXT Nloop
140   END
```

read label EXAMPLE

```
10    CLEAR SCREEN
20    MASS STORAGE IS "d:"
30    READ LABEL Id$
40    READ LABEL Id2$ FROM "c:"
50    PRINT SYSTEM$("MSI");Id$,"C:\";Id2$
60    END
```

read locator EXAMPLE

```
10    CLEAR SCREEN
20    SET LOCATOR 0,0
30    WHILE KBD$=""
40      READ LOCATOR X,Y,Stat$
50      PRINT X,Y
60      PRINT "Status: ";Stat$;" - length: ";LEN(Stat$)
70      WAIT 1
80    END WHILE
90    END
```

read EXAMPLE

```
10    DIM Array(4)
20    DATA 1, 2, 3, 4, 5
30    RESTORE
40    CLEAR SCREEN
50    READ Array(*)
60    PRINT "The data read into the array: "
70    PRINT Array(*)
80    END
```

readio EXAMPLE

```
10  ! LOAD BIN "SERIAL32"
20    LIST BIN
30    WAIT 1
40    CLEAR SCREEN
50    ON ERROR GOTO Recover
60    FOR I=0 TO 6
70      PRINT READIO(9,I);I
80      DISP I
90 Recover: DISP I
100   NEXT I
110   OFF ERROR
120   WRITEIO 9,1;2400
130   PRINT READIO(9,1)
140   END
```

real EXAMPLE

```
10    COMPLEX C
20    C=CMPLX(5,7)
30    PRINT "The real part of C is";REAL(C)
40    INTEGER A
50    A=7
60    PRINT INT(A*PI)
70    PRINT REAL(A*PI)
80    END
```

recover EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Testing the ON ERROR RECOVER statement."
30    ON ERROR RECOVER Here
40    Forceerr
50 Here: PRINT "RECOVER works properly."
60    END
70    SUB Forceerr
80      PRINT "Forcing an error"
90      PRINT SYSTEM$("WHO IS BILL GATES")  !gives error 401
100   SUBEND
```

rectangle EXAMPLE

```
10    CLEAR SCREEN
20    GINIT
30    MOVE 40,40
40    RECTANGLE 10,20
50    DISP "1"
60    WAIT 1
70    AREA PEN 3
80    RECTANGLE 10,-20,FILL
90    DISP "2"
100   WAIT 1
110   PEN 2
120   RECTANGLE -10,-20,EDGE
130   DISP "3"
140   WAIT 1
150   AREA PEN 7
160   PEN 8
170   RECTANGLE -10,20,FILL,EDGE
180   DISP "4"
190   WAIT 1
200   DISP
210   END
```

redim EXAMPLE

```
10    DIM Array(14)
20    DATA 1, 2, 3, 4, 5, 6, 8, 4, 56, 678, 678, 65, 4, 8, 6, 0, 12, 2
30    RESTORE
40    CLEAR SCREEN
50    READ Array(*)
60    PRINT "The sum of the array: ";SUM(Array)
70    REDIM Array(4:10)
80    PRINT "Array redimensioned to values 4 to 10. Sum:";SUM(Array)
90    END
```

rem EXAMPLE

```
 1     REM A REM statement is used to insert comments into programs.
 2     REM The REM statement may contain any text you wish.
 3     REM It is useful in explaining what the program is doing.
10    CLEAR SCREEN
20    REM Means, I can stick a remark here.
30    PRINT "Notice the REM statement in the code?"
40   !It works just like the "!" symbol.
50    END
```

rename EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Creating file"
30    SAVE "file.XXX"
40    CAT "*.XXX";NAMES
50    WAIT 2
60    PRINT "Now, I will rename it to"
70    RENAME "file.XXX" TO "file.ZZZ"
80    CAT "*.ZZZ";NAMES
90    WAIT 2
100   PURGE "file.ZZZ"
110   PRINT "File purged."
120   END
```

reorder EXAMPLE

```
10    OPTION BASE 1
20    DIM Matrix(3,3),Vector(3)
30    DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 3, 2, 1
40    RESTORE
50    READ Matrix(*),Vector(*)
60    CLEAR SCREEN
70    PRINT "The matrix looks like: "
80    Prtmat(Matrix(*),3,3)
90    MAT REORDER Matrix BY Vector,2
100   PRINT "MAT reorder"
110   Prtmat(Matrix(*),3,3)
120   END
130   SUB Prtmat(A(*),Lenarr,Widarr)
140! This sub prints out a matrix length of Lenarr and wide as widarr.
150! A 3x3 matrix would print like:
160!           [ 1 2 3 ]  Widarr = 3
170!           [ 4 5 6 ]
180!           [ 7 8 9 ]
190!           Lenarr = 3
200     ASSIGN @Out TO CRT
210     FOR Col=1 TO Lenarr
220       OUTPUT @Out;"  [";
230       FOR Row=1 TO Widarr
240         OUTPUT @Out;A(Col,Row);
250       NEXT Row
260       OUTPUT @Out;" ]"
270     NEXT Col
280     ASSIGN @Out TO *
290   SUBEND
```

repeat_until EXAMPLE

```
10    Count=0
20    REPEAT
30      Count=Count+2
40      PRINT Count
50    UNTIL Count=10
60    PRINT "Test done."
70    END
```

res EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Demonstrating the RES command. It returns the answer "
30    PRINT "last calculated."
40    DISP "Enter a calculation (i.e. 2+2) and then press CONT"
50    PAUSE
60    PRINT "The result was:";RES
70    END
```

re-save EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Saving code. Creating file"
30    RE-SAVE "file.txt"
40    CAT "*.txt";NAMES
50    PAUSE
60    PRINT "Now, I will purge it."
70    PURGE "file.txt"
80    CAT "*.txt";NAMES
90    PRINT "File purged."
100   END
```

reset EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** output TEST ****"
30    CREATE "test.txt",0
40    ASSIGN @File TO "test.txt";FORMAT ON
50    OUTPUT @File;"This is a test"
60    RESET @File
70    ENTER @File;Test$
80    PRINT Test$
90    ASSIGN @File TO *
100   PURGE "test.txt"
110   END
```

re-store key EXAMPLE

```
10    RE-STORE KEY "keys2"
20    READ KEY 1,Str$
30    PRINT "Key 1 was ";"EDIT";" now is QUIT -- Please type CONT"
40    SET KEY 1,"QUIT"
50    PAUSE
60    LOAD KEY "keys2"
70    PRINT "Keys are restored to normal."
80    PURGE "keys2"
90    END
```

restore EXAMPLE

```
10    DIM Array(4)
20 Here: DATA 1, 2, 3, 4, 5
30    RESTORE  !RESTORE moves the pointer back to the beginning of the DATA.
40    CLEAR SCREEN
50    FOR Loop=1 TO 3
60      READ Array(*)
70      RESTORE Here
80      PRINT "The array: "
90      PRINT Array(*)
100   NEXT Loop
110   END
```

re-store EXAMPLE

```
10    CLEAR SCREEN
20    RE-STORE "junk.XXX"
30    PRINT "Storing code"
40    PRINT "Program paused"
41    PRINT "Press CONT"
50    PAUSE
60    PURGE "junk.XXX"
70    ON ERROR GOTO 110
80    CAT "junk.XXX";NAMES
90    PRINT "The file was not purged correctly"
100   STOP
110   PRINT "File purged"
120   END
```

resume interactive EXAMPLE

```
10    X=5
20    ON TIME (TIMEDATE+X) MOD 86400 GOTO Here
30    PRINT "I'll wait";X;"seconds. Keys disabled."
40    SUSPEND INTERACTIVE
50 Loop: GOTO Loop
60 Here: RESUME INTERACTIVE
70    PRINT "Test done."
80    END
```

return EXAMPLE

```
10    CLEAR SCREEN
20    GOSUB Here
30    PRINT "Test done."
40    STOP
50 Here: PRINT "Currently in the SUB."
60    RETURN
70    END
```

rev$ EXAMPLE

```
10    DIM Test$[30]
20    Test$="This is the string to reverse"
30    CLEAR SCREEN
40    PRINT Test$
50    PRINT
60    PRINT REV$(Test$)  !Reverse the string.
70    END
```

rnd EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Yhatzee"
30    RANDOMIZE
40    FOR Loop=1 TO 5
50      PRINT "Die";Loop;":";INT(RND*6)+1
60    NEXT Loop
70    END
```

rotate EXAMPLE

```
10    INTEGER X,Y
20    X=100
30    Msg("Original bits")
40    See(X)
50    PRINT
60    Y=ROTATE(X,5)
70    Msg("ROTATE shifts the bits with wrap-around")
80    See(Y)
90    END
100   SUB See(INTEGER X)
110     FOR Loop=15 TO 0 STEP -1
120       Temp=BIT(X,Loop)
130       PRINT Temp;
140     NEXT Loop
150   SUBEND
160   SUB Msg(Str$)
170     PRINT Str$
180   SUBEND
```

rplot EXAMPLE

```
10    CLEAR SCREEN
20    GINIT
30    MOVE 40,40
40    RECTANGLE 10,20
50    DISP "1"
60    WAIT 1
70    AREA PEN 3
80    RPLOT 5,2
90    RECTANGLE 10,-20,FILL
100   DISP "2"
110   WAIT 1
120   PEN 2
130   RECTANGLE -10,-20,EDGE
140   DISP "3"
150   WAIT 1
160   AREA PEN 7
170   PEN 8
180   RECTANGLE -10,20,FILL,EDGE
190   DISP "4"
200   WAIT 1
210   DISP
220   END
```

rpt$ EXAMPLE

```
10    DIM Test$[80]
20    Test$=RPT$("*",20)
30    PRINT RPT$("=",3),Test$
40    END
```

rsum EXAMPLE

```
10    OPTION BASE 1
20    DIM Matrix(3,3)
30    DIM Vector(3)
40    DATA 1, 2, 3, 4, 5, 6, 7, 8, 9
50    RESTORE
60    READ Matrix(*)
70    CLEAR SCREEN
80    PRINT "The matrix looks like: "
90    Prtmat(Matrix(*),3,3)
100   MAT Vector=RSUM(Matrix)
110   PRINT "The RSUM vector is [";Vector(*);"]"
120   END
130   SUB Prtmat(A(*),Lenarr,Widarr)
140! This sub prints out a matrix length of Lenarr and wide as widarr.
150! A 3x3 matrix would print like:
160!          [ 1 2 3 ]  Widarr = 3
170!          [ 4 5 6 ]
180!          [ 7 8 9 ]
190!          Lenarr = 3
200     ASSIGN @Out TO CRT
210     FOR Col=1 TO Lenarr
220       OUTPUT @Out;"  [";
230       FOR Row=1 TO Widarr
240         OUTPUT @Out;A(Col,Row);
250       NEXT Row
260       OUTPUT @Out;" ]"
270     NEXT Col
280     ASSIGN @Out TO *
290   SUBEND
```

runlight EXAMPLE

```
10    CLEAR SCREEN
20    RUNLIGHT OFF
30    FOR J=1 TO 5
40      WAIT 1
50      PRINT J
60    NEXT J
70    END
```

save EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Saving code. Creating file"
30    SAVE "file.txt"
40    PRINT "Program paused."
50    PAUSE
60    PURGE "file.txt"
70    PRINT "File purged."
80    END
```

sc EXAMPLE

```
10    CLEAR SCREEN
20    ASSIGN @Out TO CRT
30    PRINT "The ISC for @Out is ";SC(@Out);"."
40    ASSIGN @Out TO *
50    END
```

scratch EXAMPLE

```
10  ! SCRATCH        !This command will clear the current program out of
memory.
20  ! SCRATCH KEY 2  !This command will clear the F2 softkey.
30  ! SCRATCH A      !This command clears all variables of the basic program
and the program also.
40  ! SCRATCH ALL    !This command is synonymous with SCRATCH A.
50  ! SCRATCH B      !This command is synonymous with SCRATCH A.
60  ! SCRATCH BIN    !This command is synonymous with SCRATCH A.
70  ! SCRATCH C      !This command clears all variable of the basic program,
but leaves the program intact.
80  ! SCRATCH COM    !This command is synonymous with SCRATCH C.
90  ! SCRATCH R      !This command clears teh keyboard RECALL buffer.
100 ! SCRATCH RECALL !This command is synonymous with SCRATCH R.
110   END
```

select_case EXAMPLE

```
10    CLEAR SCREEN
20    INPUT "Please enter your age:",Age
30    SELECT Age
40    CASE <1,>100
50      PRINT "Congratulations - Movie is free!"
60      GOTO End
70    CASE <12
80      Price=2.00
90    CASE 12 TO 59
100     Price=6.50
110   CASE 60
120     PRINT "Special movie rate"
130     Price=3.00
140   CASE ELSE
150     Price=4.50
160   END SELECT
170 Image: IMAGE "Movie price is $", D.2D
180   PRINT USING Image;Price
190 End: END
```

separate alpha EXAMPLE

```
10    MERGE ALPHA WITH GRAPHICS
20    CLEAR SCREEN
30    PRINT "With Alpha and Graphics merged, the text and"
40    PRINT "graphics should disappear with a CLEAR SCREEN"
50    PAUSE
60    MOVE 0,60
70    AREA PEN 4
80    RECTANGLE 30,30,FILL
90    WAIT 2
100   SEPARATE ALPHA FROM GRAPHICS
110   CLEAR SCREEN
120   PRINT "This text should disappear with a CLEAR SCREEN"
130   PRINT "The square should not be erased"
140   MOVE 0,60
150   AREA PEN 7
160   RECTANGLE 30,30,FILL
170   WAIT 2
180   CLEAR SCREEN
190   WAIT 1
200   PRINT "Notice the square is not erased!"
210   END
```

set echo EXAMPLE

```
10    PLOTTER IS CRT,"INTERNAL";COLOR MAP
20    CLEAR SCREEN
30    SET ECHO 65,50    !Set the crosshair to the middle of the screen.
40    READ LOCATOR X,Y  !Read where the mouse pointer is at.
50    PRINT X,Y
60    END
```

set key EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "Program shows soft key control statements."
30    STORE KEY "keys2"! store softkey definitions
40    READ KEY 1,Str$! read current key label
50    PRINT "Key 1 was ";Str$;" now is QUIT -- Please type CONT"
60    SET KEY 1,"QUIT"! change key label
70    PAUSE
80    LOAD KEY "keys2"! load original sofkey definitions
90    PRINT "Key 1 is back to normal."
100   PURGE "keys2"! delete file.
110   END
```

set locator EXAMPLE

```
10    CLEAR SCREEN
20    SET LOCATOR 0,0
30    WHILE KBD$=""
40      READ LOCATOR X,Y  !Move the mouse around.
50      PRINT X,Y
60      WAIT 1
70    END WHILE
80    END
```

set pen EXAMPLE

```
10    DATA .5,.5,.5    ! 8  = dark grey
20    DATA .75,.75,.75 ! 9  = light grey
30    DATA .5, 0, 0    ! 10 = dark red
40    DATA .5,.5, 0    ! 11 = dark yellow
50    DATA 0,.5, 0     ! 12 = dark green
60    DATA 0,.5,.5     ! 13 = dark cyan
70    DATA 0, 0,.5     ! 14 = dark blue
80    DATA .5, 0,.5    ! 15 = dark magenta
90    RESTORE
100   DIM Palette(8:15,1:3)
110   CLEAR SCREEN
120   PRINT "*** SET PEN TEST ****"
130   PLOTTER IS CRT,"INTERNAL";COLOR MAP
140   READ Palette(*)
150   SET PEN 8 INTENSITY Palette(*)
160   X=0
170   Y=85
180   FOR Loop=8 TO 15
190     MOVE X,Y
200     AREA PEN Loop
210     RECTANGLE 10,10,FILL,EDGE
220     WAIT 1
230     X=X+10
240   NEXT Loop
250   END
```

set time EXAMPLE

```
10    Current_time=TIMEDATE
20    CLEAR SCREEN
30    PRINT "Setting time to 1:00 P.M."
40    SET TIME TIME("13:00:00")
50    PRINT "Press CONT to continue..."
60    PAUSE
70    SET TIMEDATE Current_time        !Change the timedate to Current_time.
80    PRINT "The time might be off a bit."
90    END
```

set timedate EXAMPLE

```
10    Current_time=TIMEDATE
20    CLEAR SCREEN
30    PRINT "Setting time to 12:00 P.M. June 11, 1976"
40    SET TIMEDATE TIME("12:00:00")+DATE("11 Jun 1976")
50    PRINT "Press CONT to continue..."
60    PAUSE
70   SET TIMEDATE Current_time
80   PRINT "The time might be off a bit."
90   END
```

sgn EXAMPLE

```
10    CLEAR SCREEN
20    IF SGN(-87)=-1 THEN PRINT "-87 is negative"
30    IF SGN(9)=1 THEN PRINT "9 is positive"
40    IF SGN(0)=0 THEN PRINT "0 is zero"
50    END
```

shift EXAMPLE

```
10    INTEGER Y
20    Y=1000
30    CLEAR SCREEN
40    Msg("Original bits")
50    See(Y)
60    Y=SHIFT(Y,5)  !Shift the bits of Y over by 5 bits.
70    PRINT
80    Msg("Shift the bits over by 5")
90    See(Y)
100   END
110   SUB See(INTEGER X)
120     FOR Loop=15 TO 0 STEP -1
130       Temp=BIT(X,Loop)
140       PRINT Temp;
150     NEXT Loop
160   SUBEND
170   SUB Msg(Str$)
180     PRINT Str$
190   SUBEND
```

sin EXAMPLE

```
10!    This example deminstrates the usage of the trigonometric
20! functions. The following triangle will be used:
30!
40!    |\
50!    |a \        Given C = 5 units and angle c = 35 degrees
60!   C|    \B
70!    |b    c\           Note: angle b = 90 dgrees.
80!    +--------
90!        A
100   CLEAR SCREEN
110   DEG ! get in degree mode
120   REAL A,B,C
130! Given:
140   C=5.0
150   Angle_b=90
160   Angle_c=35
170! Angle a can be found by simply subtracting the total given
180! angles by 1800 degrees. Every triangle only has 180
190! degress.
200   Angle_a=180-(Angle_c+Angle_b)
210! The sine of angle c is definded as C over B. Solving for
220! B gives us:
230   B=C/SIN(Angle_c)
240! The cosine of angle c is definded as A over B. Solving for
250! A gives us:
260   A=B*COS(Angle_c)
270! To double check the answers, onte posible way is:
280! Given: A^2 + C^2 = B^2 and solving for C
290   X=SQR(B^2-A^2)
300   X=DROUND(X,1)
310   IF X=C THEN
320     PRINT "The leg A =";A;"units."
330     PRINT "The leg B =";B;"units."
340     PRINT "The leg C =";C;"units."
350     PRINT "Angle a is = ";Angle_a;"degrees."
360     PRINT "Angle b is = ";Angle_b;"degrees."
370     PRINT "Angle c is = ";Angle_c;"degrees."
380   ELSE
390     PRINT "An error has occured."
400   END IF
410   END
```

sinh EXAMPLE

```
10    COMPLEX C
20    C=CMPLX(4,7)
30    CLEAR SCREEN
40    PRINT "SINH of 80 is:";SINH(80)
50    X=REAL(SINH(C))
60    Y=SINH(REAL(C))*COS(IMAG(C))
70    PRINT "x=";X,"y=";Y
80    IF X=Y THEN
90      PRINT "True"
100   ELSE
110     PRINT "False - error in documentation."
120   END IF
130   X=IMAG(SINH(C))
140   Y=COSH(REAL(C))*SIN(IMAG(C))
150   PRINT "x=";X,"y=";Y
160   IF X=Y THEN
170     PRINT "True"
180   ELSE
190     PRINT "False - error in documentation."
200   END IF
210   END
```

size EXAMPLE

```
10    OPTION BASE 1
20    DIM A(16,6)
30    CLEAR SCREEN
40    Pass_a(A(*))
50    PRINT "After REDIM"
60    REDIM A(7,3)
70    Pass_a(A(*))
80    END
90    SUB Pass_a(REAL A(*))
100     L=RANK(A)
110     PRINT "The number of elements in each dimension of A is
(";SIZE(A,1);",";SIZE(A,2);")"
120   SUBEND
```

spanish EXAMPLE

```
10    LEXICAL ORDER IS SPANISH
20    PRINT SYSTEM$("LEXICAL ORDER IS")
30    END
```

sqr EXAMPLE

```
10! Please note SQRT and SQR are exactly the same. In fact,
20! SQRT is parced to mean SQR.
30    Number=49
40    COMPLEX Z
50    Z=CMPLX(4,7)
60    CLEAR SCREEN
70    PRINT "The squre root of ";Number;" is:";SQR(Number)
80    PRINT
90    PRINT "Testing the square root of complex numbers."
100   X=REAL(SQR(Z))
110   Y=SQR((SQR(REAL(Z)^2+IMAG(Z)^2)+REAL(Z))/2)
120   PRINT X;"=";Y
130   IF (X=Y) THEN PRINT "Check"
140   X=IMAG(SQR(Z))
150   Y=SGN(Z)*SQR((SQR(REAL(Z)^2+IMAG(Z)^2)-REAL(Z))/2)
160   PRINT X;"=";Y
170   IF (X=Y) THEN
180     PRINT "Check"
190   ELSE
200     PRINT "What a minute. What is the Y for?"
210   END IF
220   PRINT
230   Drawtriangle
240   PRINT "Given a^2 + b^2 = c^2, then c = sqr(a^2 + b^2)"
250   A=3
260   B=4
270   C=SQR(A^2+B^2)
280   PRINT "a = ";A
290   PRINT "b = ";B
300   PRINT "c = ";C
310   END
320   SUB Drawtriangle
330     PRINT " |\"
340     PRINT "a|  \ c"
350     PRINT " +++++"
360     PRINT "   b"
370   SUBEND
```

sqrt EXAMPLE

```
10! Please note SQRT and SQR are exactly the same. In fact,
20! SQRT is parced to mean SQR.
30    Number=49
40    COMPLEX Z
50    Z=CMPLX(4,7)
60    CLEAR SCREEN
70    PRINT "The squre root of ";Number;" is:";SQR(Number)
80    PRINT
90    PRINT "Testing the square root of complex numbers."
100   X=REAL(SQR(Z))
110   Y=SQR((SQR(REAL(Z)^2+IMAG(Z)^2)+REAL(Z))/2)
120   PRINT X;"=";Y
130   IF (X=Y) THEN PRINT "Check"
140   X=IMAG(SQR(Z))
150   Y=SGN(Z)*SQR((SQR(REAL(Z)^2+IMAG(Z)^2)-REAL(Z))/2)
160   PRINT X;"=";Y
170   IF (X=Y) THEN
180     PRINT "Check"
190   ELSE
200     PRINT "What a minute. What is the Y for?"
210   END IF
220   PRINT
230   Drawtriangle
240   PRINT "Given a^2 + b^2 = c^2, then c = sqr(a^2 + b^2)"
250   A=3
260   B=4
270   C=SQR(A^2+B^2)
280   PRINT "a = ";A
290   PRINT "b = ";B
300   PRINT "c = ";C
310   END
320   SUB Drawtriangle
330     PRINT " |\"
340     PRINT "a|  \ c"
350     PRINT " +++++"
360     PRINT "   b"
370   SUBEND
```

standard EXAMPLE

```
10    LEXICAL ORDER IS STANDARD
20    PRINT SYSTEM$("LEXICAL ORDER IS")
30    END
```

status EXAMPLE

```
10    PRINT STATUS(CRT,9)    !Gets the screen width.
20    PRINT STATUS(CRT,13)   !Gets the screen height.
30    PRINT STATUS(CRT,17)   !Gets the input line color.
40    END
```

step EXAMPLE

```
10    FOR J=40 TO 500 STEP 20
20      PRINT J
30    NEXT J
40    END
```

stop EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "I can stop the program before it is finished."
30    STOP
40    END
```

store key EXAMPLE

```
10    CLEAR SCREEN
20    STORE KEY "keys2"
30    READ KEY 1,Str$
40    PRINT "Key 1 was ";Str$;" now is QUIT -- Please type CONT"
50    SET KEY 1,"QUIT"
60    PAUSE
70    LOAD KEY "keys2"
80    PRINT "Key 1 returned to normal."
90    PURGE "keys2"
100   END
```

store EXAMPLE

```
10    CLEAR SCREEN
20    STORE "junk.XXX"
30    PRINT "Saving code as"
40    CAT "junk.XXX";NAMES
50    PRINT "Program paused."
60    PAUSE
70    PURGE "junk.XXX"
80    ON ERROR GOTO 120
90    CAT "junk.XXX";NAMES
100   PRINT "The file was not purged properly"
110   STOP
120   PRINT "File purged"
130   END
```

sub EXAMPLE

```
10    CLEAR SCREEN
20    Mysub
30    END
40    SUB Mysub
50      PRINT "In My SUB"
60    SUBEND
```

subend EXAMPLE

```
10    CLEAR SCREEN
20    Mysub
30    END
40    SUB Mysub
50      PRINT "In My SUB"
60    SUBEND
```

sum EXAMPLE

```
10    DIM Array(4)
20    DATA 1, 2, 3, 4, 5
30    RESTORE
40    CLEAR SCREEN
50    READ Array(*)
60    PRINT "The sum of the array: "
70    PRINT Array(*)
80    PRINT "is";SUM(Array);"."
90    END
```

suspend interactive EXAMPLE

```
10    X=5
20    ON TIME (TIMEDATE+X) MOD 86400 GOTO Here
30    PRINT "I'll wait";X;"seconds. Keys disabled."
40    SUSPEND INTERACTIVE
50 Loop: GOTO Loop
60 Here: RESUME INTERACTIVE
70    PRINT "Test done."
80    END
```

swedish EXAMPLE

```
10    LEXICAL ORDER IS SWEDISH
20    PRINT SYSTEM$("LEXICAL ORDER IS")
30    END
```

symbol EXAMPLE

```
10    DIM A(6,2)
20    DATA 1, 7, -2, 16, 7, -1, 15, 4, -1, 21, 8, -1, 15, 12, -1, 16, 9, -1,
1, 9, -1
30    RESTORE
40    CLEAR SCREEN
50    GINIT
60    READ A(*)
70    MOVE 50,50
80    PEN 6
90    AREA PEN 7
100   SYMBOL A(*),FILL,EDGE
110   END
```

system keys EXAMPLE

```
10    SYSTEM KEYS
20    END
```

system priority EXAMPLE

```
10    CLEAR SCREEN
20    SYSTEM PRIORITY 1
30    PRINT "Priority: "&SYSTEM$("SYSTEM PRIORITY")
40    END
```

system$ EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** System$ TEST ****"
30    PRINT "Memory "&SYSTEM$("AVAILABLE MEMORY")
40    PRINT "CRT ID: "&SYSTEM$("CRT ID")
50    PRINT "DISP line: "&SYSTEM$("DISP LINE")
60    PRINT "Dump devices: "&SYSTEM$("DUMP DEVICE IS")
70    PRINT "Graphics: "&SYSTEM$("GRAPHICS INPUT IS")  !401
80    PRINT "Input line is: "&SYSTEM$("KBD LINE")
90    PRINT "Lexical order: "&SYSTEM$("LEXICAL ORDER IS")
100   PRINT "Mss memory: "&SYSTEM$("MASS MEMORY")
110   PRINT "Mass storage: "&SYSTEM$("MSI")
120   PRINT "Plotter: "&SYSTEM$("PLOTTER IS")
130   PRINT "Printer: "&SYSTEM$("PRINTER IS")
140   PRINT "Process ID: "&SYSTEM$("PROCESS ID")
150   PRINT "Serial number: "&SYSTEM$("SERIAL NUMBER")   !401
160   PRINT "System ID: "&SYSTEM$("SYSTEM ID")
170   PRINT "Priority: "&SYSTEM$("SYSTEM PRIORITY")
180   PRINT "Timezone: "&SYSTEM$("TIMEZONE IS")
190   PRINT "Trig mode: "&SYSTEM$("TRIG MODE")
200   PRINT "BASIC ver: "&SYSTEM$("VERSION:BASIC")
210   PRINT SYSTEM$("VERSION:HTB")
220   PRINT "OS ver: "&SYSTEM$("VERSION:OS")
230   PRINT "Wild cards are: "&SYSTEM$("WILDCARDS")
240   PRINT "Window system: "&SYSTEM$("WINDOW SYSTEM")
250   END
```

tab EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** Print TEST ****"
30    PRINT "[TAB]";TAB(15);"15 spaces"
40    PRINT TABXY(5,5);"TABXY test"
50    END
```

tabxy EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** Print TEST ****"
30    PRINT "[TAB]";TAB(15);"15 spaces"
40    PRINT TABXY(5,5);"TABXY test"
50    END
```

tan EXAMPLE

```
10!    This example deminstrates the usage of the trigonometric
20! functions. The following triangle will be used:
30!
40!    |\
50!    |a \        Given C = 5 units and angle c = 35 degrees
60!   C|    \B
70!    |b    c\            Note: angle b = 90 dgrees.
80!    +--------
90!        A
100   CLEAR SCREEN
110   DEG ! get in degree mode
120   REAL A,B,C
130! Given:
140   C=4
150   Angle_c=60
160   Angle_b=90
170! Angle a can be found by simply subtracting the total given
180! angles by 180 degrees. Every triangle only has 180
190! degress.
200   Angle_a=180-(Angle_c+Angle_b)
210! The sine of angle c is definded as C over B. Solving for
220! B gives us:
230   B=C/SIN(Angle_c)
240! The tangent of angle c is definded as C over A. Solving for
250! A gives us:
260   A=C/TAN(Angle_c)
270! To double check the answers, one possible way is:
280! Given: A^2 + C^2 = B^2 and solving for C
290   IF SQR(B^2-A^2)=C THEN
300     PRINT "The leg A =";A;"units."
310     PRINT "The leg B =";B;"units."
320     PRINT "The leg C =";C;"units."
330     PRINT "Angle a is = ";Angle_a;"degrees."
340     PRINT "Angle b is = ";Angle_b;"degrees."
350     PRINT "Angle c is = ";Angle_c;"degrees."
360   ELSE
370     PRINT "An error has occured."
380   END IF
390   END
```

tanh EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "The TANH of 80 is:";TANH(80)  !Returns the hyperbolic tangent of 
80.
30    END
```

then EXAMPLE

```
10    IF 1 THEN
20      CLEAR SCREEN
30      PRINT 5
40      BEEP
50    ELSE
60      PRINT "NO"
70      STOP
80    END IF
90    END
```

time$ EXAMPLE

```
10    CLEAR SCREEN
20    PRINT TIME$(TIMEDATE)
30    END
```

time1 EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "There have been";TIME(TIME$(TIMEDATE));"seconds pass"
30    PRINT "since midnight"
40    END
```

timedate EXAMPLE

```
10    CLEAR SCREEN
20    PRINT DATE$(TIMEDATE),TIME$(TIMEDATE)
30    PRINT "Number of seconds since midnight:";TIMEDATE MOD 86400
40    PRINT "The number of seconds from 4713 B.C. is",FNJd(DATE$(TIMEDATE))
50    END
60    DEF FNJd(A$)
70      RETURN (DATE(A$) DIV 86400)-1
80    FNEND
```

timeout EXAMPLE

```
10    !LOAD BIN "SERIAL32"
20    ON TIMEOUT 9,1 GOTO 60
30    PRINT "WAITING..."
40    ENTER 9;X$
50    GOTO 30
60    PRINT "IT TIMED OUT"
70    END
```

to EXAMPLE

```
10    FOR J=40 TO 500 STEP 20
20      PRINT J
30    NEXT J
40    END
```

trace EXAMPLE

```
10    TRACE ALL
20    FOR J=40 TO 100 STEP 10
30      PRINT J
40      IF J=60 THEN TRACE OFF
50    NEXT J
60    END
```

track EXAMPLE

```
10    CLEAR SCREEN
20    TRACK CRT IS ON
30    DIGITIZE X,Y,Stat$   !Move the mouse around the screen.
40    PRINT "x:";X,"y:";Y  !It should be traced wherever it goes.
50    PRINT "Status: ";Stat$
60    PRINT "Track is ";Stat$[5,5]
70    IF Stat$[5,5]="1" THEN
80      PRINT "ON"
90    ELSE
100     PRINT "OFF"
110   END IF
120   END
```

transfer EXAMPLE

```
10    CLEAR SCREEN                          !This test requires two machines
connected by
20    PRINT "*** Transfer Test ****"        !serial cables.  One running this
program, and
30    DIM A$[50]                            !the other outputing information to
this one.
40    ASSIGN @Buf TO BUFFER [2000]
50    ASSIGN @In TO 9
60    TRANSFER @In TO @Buf
70    ENTER @Buf;A$
80    PRINT A$
90    GOTO 70
100  END
```

trim$ EXAMPLE

```
10    Test$=TRIM$("hello        ")  !Trim off the extra spaces.
20    PRINT "'";Test$;"' is of length";LEN(Test$)
30    END
```

trn EXAMPLE

```
10    DIM Matrix(1:3,1:3),M(1:3,1:3)
20    DATA 1, 2, 3, 4, 5, 6, 7, 8, 9
30    RESTORE
40    READ Matrix(*)
50    CLEAR SCREEN
60    PRINT "The matrix looks like: "
70    Prtmat(Matrix(*),3,3)
80    MAT M=TRN(Matrix)
90    PRINT "Transpose matrix"
100  Prtmat(M(*),3,3)
110  END
120  SUB Prtmat(A(*),Lenarr,Widarr)
130    ! This sub prints out a matrix length of Lenarr and wide as Widarr.
140    ! A 3x3 matrix would print like:
150    !           [ 1 2 3 ]  Widarr = 3
160    !           [ 4 5 6 ]
170    !           [ 7 8 9 ]
180    !           Lenarr = 3
190    ASSIGN @Out TO CRT
200    FOR Col=1 TO Lenarr
210      OUTPUT @Out;"  [";
220      FOR Row=1 TO Widarr
230        OUTPUT @Out;A(Col,Row);
240      NEXT Row
250      OUTPUT @Out;" ]"
260    NEXT Col
270    ASSIGN @Out TO *
280  SUBEND
```

unlock EXAMPLE

```
10    CLEAR SCREEN
20    PRINT "*** Unlock Test ****"
30    PRINT
40    CREATE "test.txt",100
50    DIM Test$[100]
60    ASSIGN @File TO "test.txt";FORMAT ON
70    PRINT "CAT after assignment..."
80    CAT "test.txt";NO HEADER
90    OUTPUT @File;"This is the contents of test.txt"
100   RESET @File
110   ENTER @File;Test$
120   PRINT
130   PRINT Test$
140   PRINT
150   LOCK @File;CONDITIONAL Error                         !This locks a file
while you are working on it.
160   PRINT "CAT after lock..."
170   CAT "test.txt";NO HEADER
180   PRINT
190   PRINT "Lock result is";Error
200   UNLOCK @File                                         !This unlocks the
file.
210   PRINT
220   PRINT "CAT after unlock..."
230   CAT "test.txt";NO HEADER
240   ASSIGN @File TO *                                    !This will also
unlock the file. Comment out the unlock
250   PRINT                                                !statement to show
this.
260   PRINT "CAT after file is closed..."
270   CAT "test.txt";NO HEADER
280   PURGE "test.txt"
290   END
```

until EXAMPLE

```
10    CLEAR SCREEN
20    Loop=0
30    REPEAT
40      Loop=Loop+1
50      PRINT Loop
60      WAIT 1
70    UNTIL Loop=10
80    PRINT "If the last number printed is 10, then the test passed."
90    END
```

upc$ EXAMPLE

```
10    Test$=UPC$("hello")  !Convert the string to all uppercase letters.
20    PRINT Test$
30    END
```

user keys EXAMPLE

```
10    KBD CMODE OFF
20    FOR Loop=1 TO 3
30      USER Loop KEYS
40      WAIT 2
50    NEXT Loop
60    USER 1 KEYS
70    END
```

using EXAMPLE

```
10    OPTION BASE 1
20    DIM A(3,3)
30    DATA -4, 36, 2.3, 5, 89, 17,  -6,  -12, 42, 1, 2, 3
40    RESTORE
50    !  Format (Fmt) for specified matrix (3x3)
60 Fmt3x3: IMAGE 3("[",3DD.DD,3DD.DD,3DD.DD,"]",/)
70    CLEAR SCREEN
80    READ A(*)
90    PRINT USING Fmt3x3;A(*)
100   END
```

val$ EXAMPLE

```
10    DIM A$[80]
20    A$=VAL$(34)
30    PRINT A$
40    A$=VAL$(-674)
50    PRINT A$
60    A$=VAL$(3.14)
70    PRINT A$
80    A$=VAL$(4567349765)
90    PRINT A$
100   END
```

val1 EXAMPLE

```
10    PRINT VAL("1")
20    PRINT VAL("34")
30    PRINT VAL("-674")
40    PRINT VAL(VAL$(3.14))
50    PRINT VAL(VAL$(4567349765))
60    END
```

viewport EXAMPLE

```
10    CLEAR SCREEN
20    VIEWPORT 40,80,40,80
30    FRAME
40    RECTANGLE 10,10,FILL,EDGE
50    GRID 10,10
60    END
```

wait EXAMPLE

```
10 PRINT "I'll wait 5 seconds."
20 WAIT 5
30 PRINT "I'm done"
40 END
```

where EXAMPLE

```
10    CLEAR SCREEN
20    DIM Stat$[3],Msg$[40]
30    PRINT "*** TEST ****"
40    PRINT "Returning the logical pen position."
50    WHERE X,Y,Stat$
60    PRINT "X = ";X
70    PRINT "Y = ";Y
80    IF Stat$[1,1]="1" THEN
90      PRINT "Pen is down"
100  ELSE
110    PRINT "Pen is up"
120  END IF
130  PRINT "Comma delimitor character: ";Stat$[2,2]
140  SELECT Stat$[3,3]
150  CASE "0"
160    Msg$="outside the limits"
170  CASE "1"
180    Msg$="inside the limits, but outside the viewport"
190  CASE "2"
200    Msg$="inside limits and viewport"
210  CASE ELSE
220    Msg$="junk"
230  END SELECT
240  PRINT "Clip indicator - the point is "&Msg$
250  END
```

while EXAMPLE

```
10 Good=6
20 PRINT "Count down using WHILE loop."
30 WHILE Good
40    PRINT Good
50    Good=Good-1
60 END WHILE
70 PRINT "All finished."
80 END
```

width EXAMPLE

```
10 CLEAR SCREEN
20 PRINT "Hello world, I am the computer."   !Prints sentence across screen
30 PRINTER IS CRT;WIDTH 8                    !Changes text width of screen
40 PRINT "Hello world, I am the computer."   !Prints sentence using new text
width
50 END
```

wildcards EXAMPLE

```
10 WILDCARDS OFF !Turns wildcard usage off
20 CAT "*a*.*"   !Does a catalog
30 END
```

Welcome to the HTBasic Help system. The HTBasic Help System consists of four books including; *Installing and Using Manual*, *User’s Guide*, *Reference Manual, and the Basic Plus Programming Guide/Reference Manual*. Each of these books is outlined below.

Installing and Using Manual

The *Installing and Using Manual* details the installation process and initial configuration instructions for HTBasic. HTBasic is highly configurable including loadable device drivers, customizing keyboard key assignments, defining additional LABEL characters or configuring your PC to duplicate the workstation environment. The following topics are found in the *Installing and Using Manual*:

- Installing HTBasic for Windows
- Getting Started
- GUI Description
- Using the Keyboard
- CRT and Graphic (Plotter) Drivers
- I/O Device Drivers
- Printer & Pixel Image Device Drivers
- Graphic Input Drivers
- Customizing the Environment
- Transferring Programs and Data from HP BASIC
- Changes from Earlier Releases

User's Guide

The *User's Guide* contains in-depth information about using the HTBasic language. It is arranged topically. These advanced topics will allow the user to speed program development and more fully exploit the power of HTBasic. The following topics are found in the *User’s Guide*:

- Language Elements
- Mathematics
- General Input and Output
- Files
- Serial (RS-232) I/O
- DLL Toolkit
- Program Flow Control
- Graphics
- CRT, Keyboard, and Printer
- IEEE-488 STATUS Registers
- Other I/O Destinations & Sources
- International Language Support

Reference Manual

The *Reference Manual* consists mainly of a dictionary style presentation of HTBasic keywords. Most keywords include syntax definition, samples, description of the keyword, usage, and even sample programs.

The *Reference Manual* also includes a chapter of definitions, a statement summary table, a list of error messages and an ASCII code chart. The ASCII code chart contains ASCII, decimal and hexadecimal values, and IEEE-488 commands and addresses.

Basic Plus Manuals

The Basic Plus Manuals contains detailed information for programming the Basic Plus Graphical User Interface. HTBasic Plus is a system of commands, utilities, and applications designed to enhance HTBasic programs. It provides a set of commands to create dialogs and widgets for effective graphical user interfaces.

Distributed with Release 8.3 March 2001

{ewl RoboEx32.dll, WinHelp2000, }