SylvaWare

NBASIC Version 1.5.2

User Manual

NBASIC User Manual

November 2007

This document describes the features and operation of NBASIC. It includes a detailed description of NBASIC statements and functions, as well as information about NBASIC program development.

Revision/Update Information:	This manual supersedes the *NBASIC User Manual*, Versions 1.1, 1.2, 1.2.1, 1.3, 1.4, 1.5 and 1.5.1.
Software Version:	NBASIC Version 1.5.2 (Shareware) NBASIC Version 1.5.2

SylvaWare
Evans, Georgia

Copyright © 2004, 2005, 2006, 2007 SylvaWare

NBASIC and the NBASIC logo are copyrights of SylvaWare.

All other product names mentioned herein may be trademarks of their respective companies.

SylvaWare shall not be liable for technical or editorial errors or omissions contained herein. The information in this document is provided “as is” without warranty of any kind and is subject to change without notice. The warranties for SylvaWare products are set forth in the express limited warranty statements accompanying such products. Nothing herein should be construed as constituting an additional warranty.

Contents

Preface

Intended Audience

This manual is intended for all users of NBASIC.

Document Structure

This manual is divided into two parts, each of which is subdivided into several chapters.

Part I describes the operation of NBASIC.

- Chapter 1 provides an overview of NBASIC.
- Chapter 2 describes getting started with NBASIC.
- Chapter 3 describes the following details of NBASIC:
  - File System
  - Printers
- Chapter 4 describes the user interface of NBASIC.
- Chapter 5 introduces programming with NBASIC.
- Chapter 6 provides how-to information on NBASIC.

Part II provides a reference for NBASIC.

- Chapter 7 describes the statements in NBASIC.
- Chapter 8 describes the functions in NBASIC.
- Chapter 9 describes the operators in NBASIC.
- Chapter 10 describes the control and editing keys in NBASIC.
- Chapter 11 describes the error codes in NBASIC.

Related Documents

The following documents are relevant to NBASIC:

- *NBASIC Setup Guide*
- *NBASIC Statement Reference*
- *NBASIC Function Reference*

For additional information about NBASIC, access the following World Wide Web address:
`http://sylvaware.home.mindspring.com`

Reader's Comments

SylvaWare welcomes your comments on this manual. Please send comments to the following address:

Internet: sylvaware@mindspring.com

Conventions

The following conventions are used in this manual:

Ctrl+x	A sequence such as Ctrl+x indicates that you must hold down the key labeled Ctrl while you press another key.
ENTER	In examples; a key name enclosed in a box indicates a key on the keyboard.
[ ]	In statement and function descriptions, brackets indicate optional choices. You can choose the item or not. Do not type the brackets as part of the statement or function.
|	In statement and function descriptions, vertical bars separate choices within braces; at least one choice is required. Do not type the vertical bars as part of the statement or function.
{ }	In statement and function descriptions, braces indicate required choices; you must choose one of the items listed. Do not type the braces as part of the statement or function.
bold text	This typeface represents a statement or function.
italic text	Italic text indicates arguments or parameters to statements or functions. It also indicates a variable name or user-defined function.
`Monospace text`	Monospace type indicates code examples and screen displays.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Non-decimal radixes-binary, octal, or hexadecimal-are explicitly indicated.

Part I

NBASIC Operation

Part I provides an overview of the features and operation of NBASIC. It includes an introduction to using NBASIC and a user interface guide. Part I also contains an introduction to programming with NBASIC.

1
Overview

1.1 Introduction

NBASIC is a BASIC interpreter designed to mimic the operation of 8/16bit microcomputers of the early 1980's like the Tandy Color Computer®, Apple][e®, Commodore 64®, Texas Instruments TI-99/4A®, and others. The operating system that also included the BASIC interpreter was usually stored in ROM and started immediately upon boot up.

NBASIC works much the same way and is essentially a small operating system that handles memory management, input/output, file management, and command processing. The interface to the operating system is through the BASIC interpreter. To control NBASIC you enter commands in the form of BASIC statements. NBASIC interprets the command and performs the requested operation.

1.2 Versions

NBASIC is available in two versions, shareware and standard. The shareware version is free and does not include some of the features found in the standard version such as graphics, printing, advanced editing, debugging, and configuration options.

All programs written with the shareware version of NBASIC will run on the standard version. Programs written with the standard version of NBASIC will run on the shareware version if statements and functions found only in the standard version are not used.

1.3 What's New

Multi-line edit capability
The **EDIT** statement has been updated to allow multiple lines to be edited.

Direct file input and output
Load and save strings directly to and from files.

File and volume information
Get detailed information about any file or volume.

Find text in program
Search for text in a program.

Decision functions
There are several new functions to perform conditional data selection.

String functions
There are several new functions to manipulate string data.

Additional statements and functions
Many new statements and functions have been added and documented.

New PEEKs and POKEs
Many new peeks and pokes have been added and documented.

1.4 History

Version 1.5.2 (Release 10)

ARC statement.
DLOAD and **DSAVE** statements.
FIND statement.

BEGINSWITH and **ENDSWITH** functions.
BREAK function.
CDN and **CHN** functions.
CHOOSE and **CHOOSE$** functions.
COMP and **COMPI** functions.
EMPTY function.
ENCLOSE$ function.
EVAL function.
EXTRACT$** and **REMAIN$ functions.
FILEINFO$** and **VOLINFO$ functions.
FINDONEOF function.
FLOOR function.
HYPOT function.
IIF and **IIF$** functions.
MATCH function.
MOD function.
PCOL and **PROW** functions.
SPAN and **CSPAN** functions.
SQUEEZE$ function.
SWITCH and **SWITCH$** functions.

Updated

EDIT statement.
RENAME statement.
UNLOAD statement.

CHR$ function.
LTRIM$**, **RTRIM$ and **TRIM$** functions.
MAX and **MIN** functions.

Fixed a bug to correctly initialize runtime to enable the CREATE statement.

Version 1.5.1 (Release 9)

CREATE statement.
FONT statement.
FORMAT statement.
FRAME statement.
FRE statement.
LINES statement.
LOADC and **SAVEC** statements.
OLD statement.
PUSH statement.
RUNR statement.
SPLITNAME statement.

COMPRESS$ function.
COPYSIGN function.
DOLLAR$ function.

DTR function.
FONT function.
INYN$ function.
MAKENAME$ function.
PROMPT$ function.
QUOTE$ function
RTD function.
SELECT function.
SET$ function.

Shortcuts
CREATE statement (**CR.**).

Fixed a bug to correctly parse file name extensions.
Fixed a bug to correctly output text to print zones.

Version 1.5 (Release 8)
APPEND statement.
ATTACH and **DETACH** statements.
NOINVERSE statement.
NOREVERSE statement.
RENUMBER statement.
REVERSE statement.
UNNUM statement.

BUFSIZ function.
VARPTR and **VARPTR$** functions.

Wildcards
APPEND statement.
ATTRIB statement.
COPY statement.
KILL statement.
MOVE statement.
RENAME statement.

Shortcuts
APPEND statement (**A.**).
ATTRIB statement (**AT.**).
BACKUP statement (**B.**).
COPY statement (**C.**).
KILL statement (**K.**).
HELP statement (**H.**).
LOAD statement (**L.**).
MOVE statement (**M.**).
MERGE statement (**ME.**).
RENAME statement (**R.**).
SAVE statement (**S.**).
VOLUME statement (**V.**).

Tandy/Radio Shack Color Computer 3 compatibility
HCIRCLE statement.
HCLS statement.
HCOLOR statement.
HDRAW statement.
HGET and **HPUT** statements.

HLINE statement.
HPAINT statement.
HSET and **HRESET** statements.

ERLIN function.
ERNO function.

NBASIC includes Tandy/Radio Shack Color Computer 3 graphics commands that are compatible with Super Extended Color BASIC in syntax only and are provided for ease of porting Color Computer 3 programs to NBASIC. The commands are implemented using the NBASIC graphics system and DO NOT provide hardware compatibility with the Tandy/Radio Shack Color Computer 3.

Commodore 64 compatibility
CLR statement.
CMD statement.
CONCAT statement.

Texas Instruments TI-99/4a compatibility
DISPLAY statement.

BASICA compatibility
RETURN statement.
SYSTEM statement.

SADD function.

The **DRAW** statement has been updated with additional commands.

The LOG statement has been changed to buffer output to the log file.

The interpreter now recognizes 2 double quotes in a string as a single double quote.

Added a PEEK to check if a file is attached to screen output.
Added a POKE to flush the log file.
Added a POKE to enable/disable the renumbering of only numbered lines.

Fixed a bug to correctly get the name of the default printer on startup.
Fixed a bug in the CHKUL statement to correctly check partially numbered programs.
Fixed a bug in the DIR statement to correctly use wildcards.
Fixed a bug in file and log output to write partial buffer if buffer causes file to exceed maximum size.

Version 1.4 (Release 7)
ASAVE and **BSAVE** statements.
AT statement.
BKOFF and **BKON** statements.
DIRR statement.
LOCK and **UNLOCK** statements.
POP statement.
QUIT statement.
SCREEN BACKUP and **SCREEN RESTORE** statements.
SELECT statement.

COLOR function.
COPYRIGHT$ function.

CSRLIN function.
LOWER$** and **UPPER$ functions.
LSET$** and **RSET$ functions.
PPOINT function.
RCP function.
SCREEN function.
SPLITNAME$ function.

Apple][e compatibility

CATALOG statement.
COLOR statement.
GR statement.
HCOLOR statement.
HGR statement.
HGR2 statement.
HLIN and **VLIN** statements.
HPLOT statement.
HTAB and **VTAB** statements.
INVERSE statement.
NORMAL statement.
NOTRACE statement.
PLOT statement.
TEXT statement.

SCRN function.

NBASIC includes Apple][e graphics commands that are compatible with AppleSoft BASIC in syntax only and are provided for ease of porting Apple programs to NBASIC. The commands are implemented using the NBASIC graphics system and DO NOT provide hardware compatibility with the Apple][e. The AppleSoft commands GR, HGR and HGR2 have the same graphics resolution (as provided by NBASIC) and internally call the NBASIC statement GRAPH ON. The AppleSoft commands HLIN, VLIN, PLOT, HPLOT, COLOR and HCOLOR are available regardless if GR, HGR or HGR2 are used and use the NBASIC graphics extents.

Texas Instruments TI-99/4a compatibility

BYE statement.
CALL CLEAR, **CALL SCREEN** and **CALL VCHAR** statements.
CON statement.
CONTINUE statement.
DELETE statement.
NUMBER statement.
RESEQUENCE statement.
TRACE statement.
UNBREAK statement.
UNTRACE statement.

CHRS$ function.
SEG$ function.

The LOG statement has been changed to use "log" as the default extension.
The LSET and RSET statements have been changed to use string array variables.

Improved the REMARK statement to detect which case to use when inserting the REM statement.

Changed the RND function so that RND(1) returns a number between 0 and 1.

Fixed a bug in the AUTO ON statement to correctly find the next line number.
Fixed a bug in the BREAK and NOBREAK statements to correctly use labels.
Fixed a bug in the CHAIN statement to correctly back up the program and to use labels.
Fixed a bug in the CHAIN, LOAD and RUN statements to correctly report an error if the program cannot be loaded or run.
Fixed a bug in the DUMP and LOAD statements to check the log file size.
Fixed a bug in the LOG statement to write the file buffer up to the maximum file size.
Fixed a bug in the STEP debugging statement to correctly step over IF/THEN/ELSE/END IF statements.

Version 1.3 (Release 6)

REMARK and **UNREMARK** statements.
SWAP statement.
TYPE statement.

Added a POKE to restore a program erased by NEW.

Fixed a bug in the LOAD and LOADR statements to confirm load only once.

Supports Windows XP control styles.

Version 1.2.1 (Release 5)

USING$ function.
ISNEG and **ISPOS** functions.

Version 1.2 (Release 4)

LINE EDIT statement.
OPTION BASE and **OPTION EXPLICIT** statements.
PCP and **PRINTERS** statements.

KILL # statement.
PAUSE statement.

$PRINT and **$XREF** statements.

LBOUND, **UBOUND**, **SIZE** and **MAXSIZE** functions.
CTRL$** and **SHIFT$ functions.
IS0 and **ISEMPTY$** functions.

MAX and **MIN** functions.
MAXLEN function.

The **LET** statement has been updated to allow multiple variables in assignments and multiple assignments.

The **DEL**, **LIST**, **CHKIO**, **CHKSW**, **CHKSYN**, **CHKUL**, **PROFILE**, **BREAK** and **NOBREAK** statements have been extended to support multiple line sequences.

The **LOAD** and **CHAIN** statements have been improved when executing in program mode to check if the current program is modified before loading another program.

Always Confirm New option in Editor tab (Options dialog box).

OEM and international character sets supported.

Added a **PEEK** that returns the zone width.

Fixed a bug in the **LOG** statement to check the log file size.
Fixed a bug in the **ROUND** function to correctly round numbers with fractions ending in 5.
Fixed a bug in the parser to correctly reset references when a syntax error is found.
Fixed a bug in the screen and printer drivers to correctly display text with fonts that have variable width characters.

Version 1.1 (Release 3)

LLIST statement.
OPEN PRINTER statement.
CLOSE PRINTER statement.
LPRINT and **LPRINT USING** statements.
PRINTER and **PRINTER?** statements.

LPOS function.
PRINTER$ function.

PRINT, **PRINT USING**, and **WRITE** statements updated to use printer file number (-2).

APPEND, **COPY**, and **MOVE** editing statements
FILES statement.
HOME statement.
NAME statement.
REOPEN statement.
REWIND statement.
TRUNCATE statement.
UNLOAD statement.

ACCESS function.
ATTRIB$ function.
ESC$** and **NUL$ functions.
FILEMODE, **FILEMODE$**, and **FILENAME$** functions.
FRE function.
GETALNUM$**, **GETALPHA$, **GETDIGIT$**, and **GETYN$** functions.
MAXNUM and **MINNUM** functions.
OPEN function.
POS function.
TWOPI function.

Added a **PEEK** that returns the number of printer columns.
Added a **PEEK** that indicates if a printer is open.
Added a **PEEK** that indicates if a printer is available.
Added a **POKE** to perform a formfeed or linefeed on the printer.

Fixed a bug to return to text mode when a program error occurs while in graphics mode.
Fixed a bug in the **LOF** function to correctly return the number of records in a file opened for random access.
Fixed a bug that does not report a log not open error when logging is off.

Version 1.0.1 (Release 2)

Added a **PEEK** that indicates a program is being run as the startup program specified in the startup options dialog.
Added a **POKE** that tells the **KILL** statement to use the recycle bin.

Fixed a bug in the parser to correctly parse the **COLOR** function when used in a **PRINT** statement.
Fixed a bug in the EXEC &HDA83 call to correctly change the case of compiled data statements.

Version 1.0 (Release 1)

Initial release.

2
Getting Started

2.1 Starting NBASIC

To begin using NBASIC, click on the Windows taskbar Start button, select Programs and then select NBASIC.

The NBASIC application window is divided into several elements. The frame allows you to resize the main window. The status bar provides information about the state of NBASIC. Inside the main window is the BASIC screen and may have a border if the main window is larger.

When you start NBASIC, you should see something similar to the following:

```
NBASIC Version 1.5.2
Copyright (C) 1998-2007 SylvaWare
All rights reserved

Ready
```

This displays information about what version of NBASIC you are using. Ready is a prompt that NBASIC uses to tell you that it is ready to accept input. A blinking square cursor indicates where the next character typed will be inserted.

2.2 Entering Commands

Statements are run in one of two modes, immediate or program. Immediate mode refers to statements entered when NBASIC is at the Ready prompt. In immediate mode anything you type followed by pressing the ENTER key will be executed by NBASIC. Program mode refers to statements that are entered with a preceding line number and added as part of a program. These statements are executed when the **RUN** statement is used to run the program.

You can enter NBASIC statements in immediate mode and they will be executed immediately. Enter the NBASIC statement **CLS** and press the ENTER key. It does not matter if you use upper or lower case, NBASIC is not case sensitive. The BASIC screen is cleared and the Ready prompt displayed at the top of the screen. The **CLS** statement clears the screen. If you make a mistake entering the command, `?Syntax error` will be displayed followed by what caused the error. NBASIC is telling you that it did not understand what you typed. Retype the command and press ENTER.

Enter the statement `PRINT 2+2` and press the ENTER key. NBASIC displays 4 followed by the Ready prompt on the next line. The **PRINT** statement prints information to the screen. In this example, the information given to **PRINT** is two numbers with a + between them. NBASIC adds the two numbers together and gives the result (4) to **PRINT**, which displays it on the screen. You can use NBASIC as a calculator using +, -, * (multiplication) and / (division).

Enter the statement `PRINT "2+2"` including the quotation marks and press the ENTER key. NBASIC displays 2+2 and then the Ready Prompt. Anything in quotation marks is considered a string and is printed literally.

2.3 Editing Statements

Use the **EDIT** statement to edit lines in a program. Type `EDIT` followed by the line number of the line to edit and press the ENTER key. Use the editing keys to make changes to the line. Press the ENTER key to save changes or the BREAK key to discard changes.

2.4 Writing Programs

To begin writing a program, delete any program currently in memory. Type `NEW` and press the ENTER key (if you want to keep any program currently in memory, be sure to save it before typing **NEW**).

In this example, the program will prompt for a name and then print a greeting based on the time of day. Use the input statement to prompt for a name and to read the name entered from the keyboard.

Enter the following program line at the Ready prompt and press the ENTER key:

```
10 INPUT "Name";N$
```

Line 10 uses the **INPUT** statement to print a prompt ("Name") followed by a question mark (?) and to read the name into the string variable *N$*. The name will be used in printing the greeting.

Enter the next line:

```
20 LET H=HOUR(NOW)
```

Line 20 gets the current time using the **NOW** function, then the hour of the time returned by the **NOW** function using the **HOUR** function, and assigns the hour value to the numeric variable *H*. The hour will be used to determine what time of day it is and the appropriate greeting to print.

Enter the following lines:

```
30 IF H<12 THEN PRINT "Good morning ";N$
40 IF H>=12 AND H<18 THEN PRINT "Good afternoon ";N$
50 IF H>=18 THEN PRINT "Good evening ";N$
```

Line 30 checks the hour in the variable *H* if it is before noon and if so prints the greeting "Good morning" followed by the name in the variable *N$* entered in line 10. Line 40 checks if the hour is after noon and before 6PM (time uses 24hour clock) and if so prints the greeting "Good afternoon" followed by the name. Line 50 checks if the hour as after 6PM and if so prints the greeting "Good evening" followed by the name.

Enter the next line:

```
60 END
```

Line 60 ends the program.

To run the program, use the **RUN** statement. Type `RUN` and press the ENTER key. The program will run beginning with line 10 which will display "Name?" followed by a blinking

cursor. Enter your name and press the ENTER key. One of the three greetings in lines 30 through 50 will be displayed. The program will then end and the Ready prompt displayed.

If any syntax errors occur in the program, they will need to be corrected. To correct a line, use the **EDIT** statement. Type `EDIT` followed by the line containing the error and press the ENTER key. Use the editing keys to make changes in the line and press the ENTER key to save the changes.

Notice in the status bar that MOD is displayed, this indicates the current program has been modified and has not been saved.

2.5 Saving Programs

Saving a program stores the program on disk so that it is not lost when you exit NBASIC and allows you to retrieve the program later.

To save a program, use the **SAVE** statement. A name must be given to each program, which is used to identify the program on the volume where it is saved. Type `SAVE "GREETING"` and press the ENTER key. This will save the program as "GREETING.NBA" on the default volume in binary format. If no extension is specified in the file name, ".NBA" is used. To save the program on a volume other than the default volume include the volume as part of the file name (e.g. "HOME:GREETING"). You can also specify any extension. By convention ".NBA" is used for programs saved in NBASIC's binary format and is used as the default extension by other statements. A program can also be saved in ASCII format, which is a standard text file that can be edited by other programs. To save a program in ASCII format append ,A to the end of the **SAVE** statement after the file name. Type `SAVE "GREETING.TXT",A` and press the ENTER key. This will save the program as "GREETING.TXT" on the default volume in ASCII format. Once the program has been saved the modified indicator (MOD) on the status bar is cleared indicating that the current program has not been modified.

Use the **DIR** statement to get a list of files on the volume. Type `DIR` and press the ENTER key. If you saved the program in the example you should see the program listed.

2.6 Loading Programs

Loading a program retrieves a program that was previously saved from disk.

To load a program, use the **LOAD** statement. A name must be included to identify which program to load. Type `LOAD "GREETING"` and press the ENTER key. This will load the program "GREETING.NBA" from the default volume. If no extension is specified in the file name, ".NBA" is used. To load the program from a volume other than the default volume include the volume as part of the file name (e.g. "HOME:GREETING"). You do not need to tell the **LOAD** statement what format the program is saved in as it can load programs in binary or ASCII format, which it determines from the file itself. The **LOAD** statement replaces any program in memory with the one from disk so make sure you save any program you are working on before loading another.

2.7 Running Programs

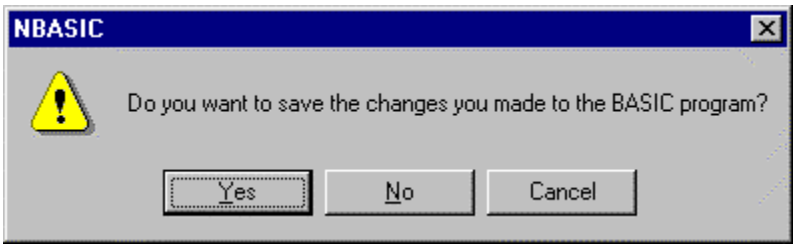
Running a program executes the statements in the program.

To run a program, use the **RUN** statement. Type `RUN` and press the ENTER key. The current program will begin executing at the lowest numbered line. The CMD indicator on the status bar will change to PRG indicating a program is running. To stop a program that is running press the BREAK key (Ctrl+Break). If the BREAK key has been disabled in the program by the **BREAK** statement, you can stop the program by selecting Reset from the context menu. You can also stop a program from running by using the **STOP** statement. Insert the **STOP** statement anywhere in the program where you wish it to stop. You can continue running the program using the **CONT** statement if the program was stopped by the BREAK key or a **STOP** statement. Execution of the program will continue with the statement where the program was stopped. If an **END** statement is executed in a program the program will end. A program will also end if there are no more lines in the program to be executed.

2.8 Exiting NBASIC

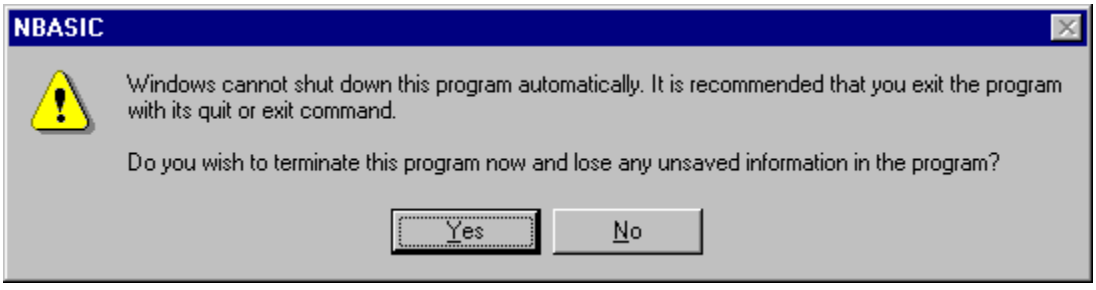
You can exit NBASIC in a number of ways. In immediate mode use the **EXIT** statement. Type `EXIT` and press the ENTER key. You can also use the standard Windows controls for closing NBASIC.

If the current program in NBASIC has been modified and has not been saved, NBASIC will prompt you with the following message:



To save the program, select Yes and you will be prompted to save the program. If you do not want to save the program select No and NBASIC closes without saving the program. To return to NBASIC without closing, select Cancel.

If you try to close NBASIC while a program is running, NBASIC will prompt you with the following message:



To continue closing NBASIC, select Yes. If the program is modified and has not been saved, you will lose the changes you have made to the program. Select No to cancel closing and to return to NBASIC.

3
Details

3.1 File System

NBASIC's file system also mimics the file systems of early microcomputers, which usually had one or two floppy disk drives. Each floppy disk contained files that were referenced by the disk drive number. There was no hierarchical arrangement of files meaning that there were no directories or folders. NBASIC references its files by using volumes, which are mapped to directories or folders on the host computer's file system. Volumes are managed through the Volumes dialog box.

A file has three components in its file name: a volume, the name, and an extension and has the format *volume*:*name*.*ext*. The volume is separated from the file name using a colon. The extension follows a period.

The volume specifies in which NBASIC volume the file is stored. If the volume is not specified in the file name the default volume is used.

The name of the file can be any valid Windows file name.

The extension is used to indicate what type of information may be in the file. Some common extensions are "dat" for data and "txt" for text. NBASIC programs saved in binary format typically use the extension "nba" while programs saved in ascii format use "bas". The extension may be omitted in many NBASIC statements that operate on files. For example, the **LOAD** statement, which is used to load programs, uses the extension "nba" if it is not specified in the file name. To load a program with another extension, you must specify the extension in the name of the program file you wish to load.

To get a list of files stored in a volume use the **DIR** statement. Type `DIR` with no argument to list all of the files on the default volume. To list the files on another volume type `DIR "`*`volume`*`:"` where *volume* is the name of the volume (note the colon). You can also use wildcards. Type `DIR "*.nba"` to list all the files with the extension "nba".

To get a list of volumes use the **VOLUMES** statement.

To change the default volume, use the **VOLUME** statement. Type `VOLUME "`*`volume`*`"` to change the default volume to *volume*. The default volume is displayed in the NBASIC status bar.

3.2 Printers

NBASIC can use any printer installed on the host operating system and provides a line printer interface independent of the type of printer being used.

When a printer is opened, NBASIC creates a print job and all output to the printer is sent to the Windows print spooler. Then, when the printer is closed, NBASIC ends the print job and the Windows print spooler begins printing the document. NBASIC will report out of memory or disk space errors returned by the print spooler during spooling, however, once the printer is closed in NBASIC, the print spooler will report any errors encountered printing the document.

When the print spooler begins to print the document, the main NBASIC window will momentarily lose the input focus.

The **LLIST** and **OPEN PRINTER** statements use the most recent printer set by the **PRINTER** or **PRINTER?** statements or if not set, the current printer specified in the Print Setup... or Print... dialogs.

To set the current printer, use the **PRINTER** statement and specify the name of the printer (as it appears in the Windows printer list). Use `PRINTER=""` to use the current printer specified in the Print Setup... or Print... dialogs.

To print the printer currently being used, use the **PCP** statement. To list the currently available printers, use the **PRINTERS** statement.

Print jobs created by the **OPEN PRINTER** statement are listed in the print manager with the document name "NBASIC". Print jobs created by the **LLIST** statement appear in the print manager with the document name "LLIST" followed by the line numbers if specified.

NBASIC will print to a printer even if it is turned off, as the print spooler manages the actual printing and will report the errors.

4
User Interface

4.1 Context Menu

The context menu provides access to the NBASIC application menu. To access the context menu right click with the mouse anywhere within the NBASIC client window.

If NBASIC is at the command prompt, the following context menu is displayed:

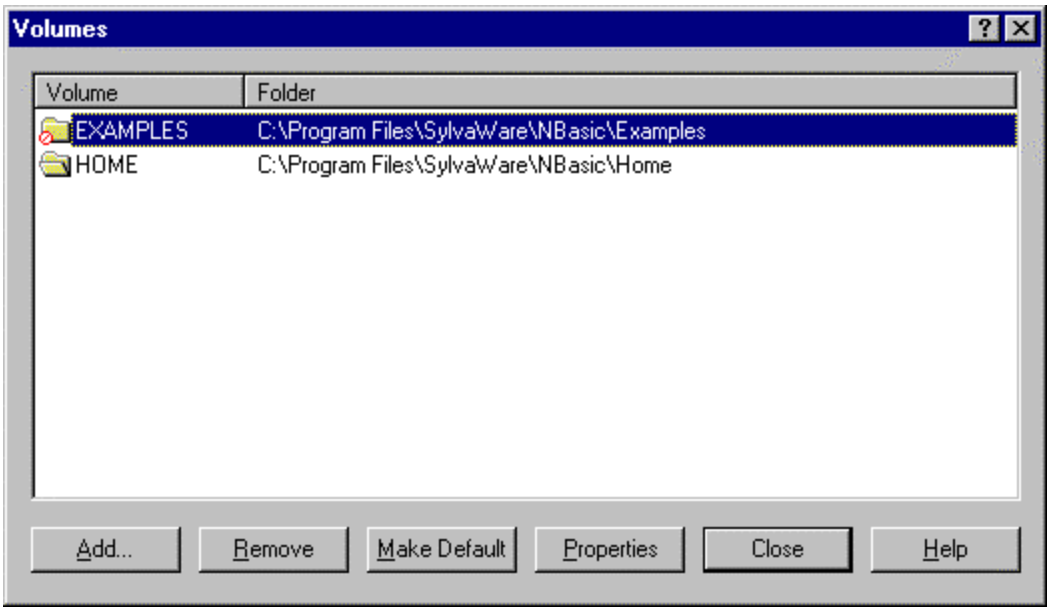
Otherwise, if NBASIC is executing a program or immediate statement the following context menu is displayed:

Context menu options:

Print Setup...	Changes the printer and printing options.
Print...	Prints the current program.
Volumes...	Manages NBASIC volumes.
Font...	Changes the NBASIC font size.
Resize	Resizes the window to fit the BASIC screen.
Reset	Resets NBASIC.
Help Topics	Lists Help topics.
About...	Displays program information, version number, and copyright.

4.2 Volumes Dialog

Use this dialog box to add, remove, and edit NBASIC volumes. To access the Volumes dialog right click with the mouse anywhere within the NBASIC client window, then select Volumes... from the context menu.



NBASIC uses volumes to store program files and data. Each volume is mapped to a Windows folder. You can create as many volumes as you need to organize your files. A volume can consist of up to sixteen alphanumeric characters. Volumes cannot be nested; they are not hierarchical like Windows folders. NBASIC maintains a default volume and is displayed on the status bar. When specifying a file with no volume, the default volume is used. Volumes can also be read-only. NBASIC cannot create, modify, or remove files on read-only volumes. Volumes can be mapped to the same Windows folder; any changes to one volume are reflected in other volumes mapped to the same Windows folder.

The NBASIC volumes and the mapped Windows folders are shown in the list. An open folder icon indicates the default volume. A folder icon with a red circle and line through it indicates a read-only volume. A red folder icon indicates that the mapped Windows folder does not exist. Click on Properties to change the volume's folder.

It is recommended that you create volumes that map to folders in the same folder where you installed NBASIC and that you give the volume the same name as the folder you are mapping.

If you rename or move a folder that is mapped to a volume, remember to update the volume's properties.

The Volumes dialog has the following options:

Add...
Adds a new volume and maps a Windows folder to it.

Remove
Removes the selected volume and deletes the mapping to the Windows folder. NBASIC will no longer have access to the files in the Windows folder mapped to this volume.

Make Default
Makes the selected volume the default volume.

Properties...
Edits the properties of the selected volume including the volume name, the mapped Windows folder, the default and read-only attributes.

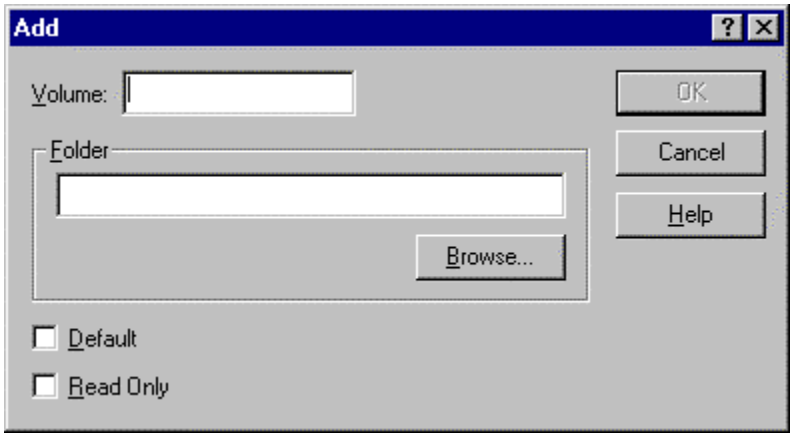
Close
Closes the Volumes dialog box.

Help
Displays help for the Volumes dialog box.

4.3 Add Dialog (Volumes)

Use this dialog to add a new volume and map a Windows folder to it.

To display this dialog, click **Add...** in the Volumes dialog.



The Add dialog has the following options:

Volume:
Enter the name of the volume to create. Volume names are limited to sixteen alphanumeric characters.

Folder
Enter the Windows folder to map to the volume.

Browse...
Browses for and selects a Windows folder to map to the volume.

Default
Specifies if the volume is to be the default volume. If checked the volume will be the default volume.

Read Only
Specifies if the volume is to be read-only. If checked the volume will be read-only. Files in read-only volumes cannot be modified or deleted and new files cannot be created.

OK
Closes the dialog box and creates the volume and maps the Windows folder to the volume.

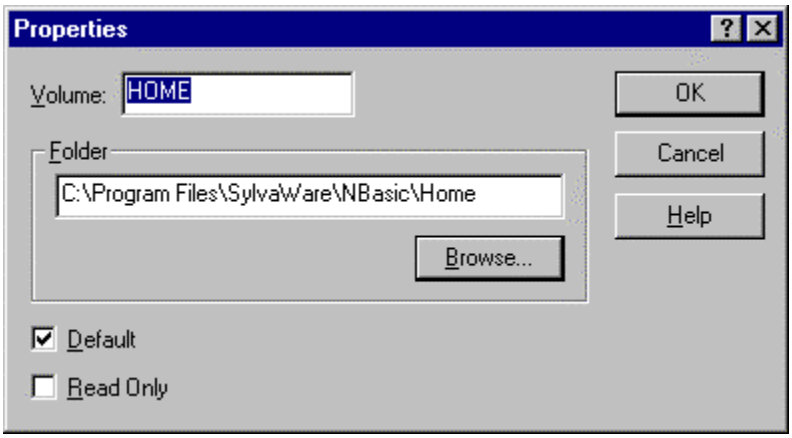
Close
Closes the dialog box without creating the volume.

Help
Displays help for the Add dialog box.

4.4 Properties Dialog (Volumes)

Use this dialog to change an existing volume's properties.

To display this dialog, click **Properties...** in the Volumes dialog.



The Properties dialog has the following options:

Volume:
Edit the name of the volume.

Folder
Change the Windows folder to map to the volume.

Browse...
Browses for and selects a Windows folder to map to the volume.

Default
Specifies if the volume is to be the default volume. If checked the volume will be the default volume.

Read Only
Specifies if the volume is to be read-only. If checked the volume will be read-only. Files in read-only volumes cannot be modified or deleted and new files cannot be created.

OK
Closes the dialog box and saves the volume's new properties.

Close
Closes the dialog box without changing the volume's properties.

Help
Displays help for the Properties dialog box.

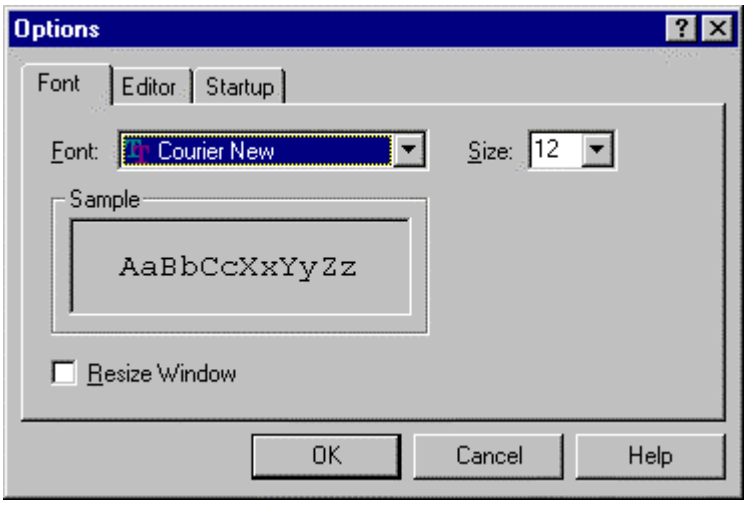
4.5 Options Dialog

Use this dialog box to configure NBASIC environment options. To access the Options dialog right click with the mouse anywhere within the NBASIC client window, then select Options... from the context menu.

There are three tabs available:

Font

Use this tab to change the text font and size.



The Font tab has the following options:

Font:

Lists the available fonts. Select the font from the drop-down list.

Size:

Lists the available font sizes. Type the font size in the box or select it from the drop down list.

Sample

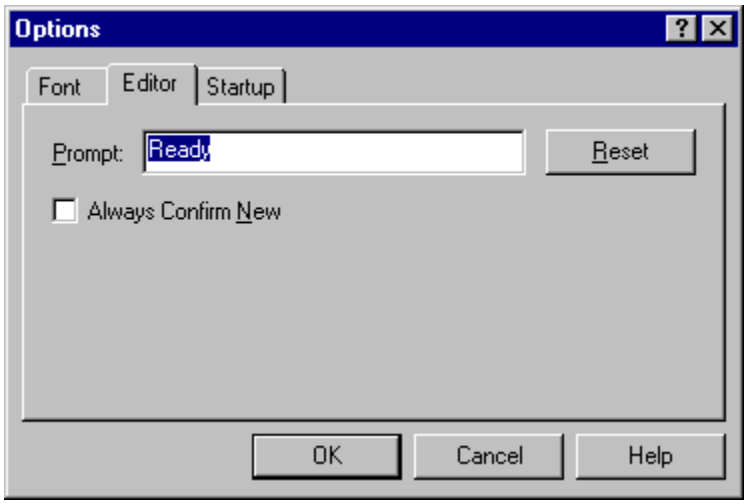
Displays an example of the selected font options.

Resize Window

Specifies if the window is to be resized to fit the NBASIC screen. If checked the NBASIC application window will be resized so that the NBASIC screen fits exactly within the window.

Editor

Use this tab to change the prompt displayed in the editor.



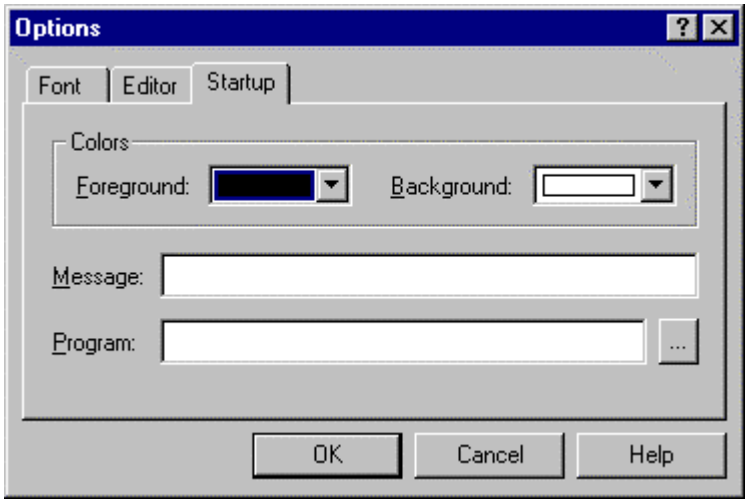
The Editor tab has the following options:

Prompt:
Enter the prompt to be displayed by NBASIC.

Reset
Resets the prompt to the original default prompt.

Always Confirm New
Specifies if the **NEW** statement is always confirmed. If checked a prompt is displayed to confirm the execution of the **NEW** statement. Clicking Yes at the prompt executes the **NEW** statement erasing the current program; clicking No does not execute the **NEW** statement. This option can be used to prevent the accidental deletion of the current program.

Startup
Use this tab to change the startup text foreground and background colors, message, and program.



Foreground:
Specifies the default startup foreground color. Select the color from the drop-down list.

Note: This does not change the current foreground color.

Background:
Specifies the default startup background color. Select the color from the drop-down list.

Note: this does not change the current background color.

Message:
Specifies a message to be displayed after startup. Enter a message up to eighty characters.

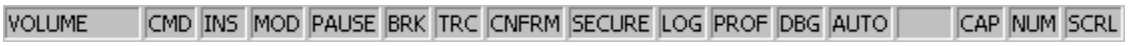
Program:
Specifies a NBASIC program to run after startup. Enter a NBASIC program or click **...** to select one.

Tip: Always include the volume name because the default volume may change.

...
Selects a NBASIC program.

4.6 Status Bar

The status bar displays information about the current state of the NBASIC environment.



Status Bar indicators:

VOLUME	Displays the default volume.
CMD	Displays CMD if NBASIC is at the command prompt, EDT if executing the **EDIT** statement, PRG if executing a program or IMM if executing an immediate statement.
INS	Displays INS if in insert mode or OVR if in overstrike mode.
MOD	Displays MOD if the current program has been modified.
PAUSE	Displays PAUSE if the PAUSE key is pressed.
BRK	Displays BRK if BREAK key trapping is on (**BREAK**).
TRC	Displays TRC if tracing is on (**TRACE**).
CNFRM	Displays CNFRM if confirmation is on (**CONFIRM**).
SECURE	Displays SECURE if secure mode is on (**SECURE**).
LOG	Displays LOG if logging is on (**LOG**).
PROF	Displays PROF if profiling is on (**PROFILE**).
DBG	Displays DBG if debugging is on (**DEBUG**).
AUTO	Displays AUTO if automatic line numbering is on (**AUTO**).
CAP	Displays CAP if Caps Lock is on.
NUM	Displays NUM if Num Lock is on.
SCRL	Displays SCRL if Scroll Lock is on.

5
Programming Guide

5.1 Arrays

Arrays are variables that contain more than one value.

Array names can be up to thirty-one (31) characters, may contain letters or digits but must begin with a letter and are case insensitive. Array names ending with a dollar sign ($) are string arrays and arrays that do not end in a dollar sign are numeric. Array names cannot begin with the names of statements or built-in functions.

Arrays are created when they are first used but must be specified with less then three (3) dimensions and ten (10) elements per dimension. To create arrays with larger numbers of elements and dimensions, use the **DIM** or **REDIM** statements. If the **OPTION EXPLICIT** statement has been specified in a program, arrays are not automatically created and must be created with the **DIM** statement.

When arrays are created, the first element of the array is at index 0. To change the base of the array to 1, use the **OPTION BASE** statement.

Numeric arrays are initialized to zero (0) while string arrays are initialized to an empty string.

To access the elements of an array, specify the index or indices of the element in parentheses.

To clear an array to its default values for each element, use the **CLEAR** statement. To delete an array variable, use the **ERASE** statement.

5.2 Branching

Execution of a program usually proceeds sequentially from line to line. To branch to another line, use the **GOTO** statement. To conditionally execute statements or branch to another line, use the **IF** statement.

The following example program prompts for a number between 1 and 10 and then prints whether or not the number is odd or even:

```
10 INPUT "Enter a number between 1 and 10 or 0 to quit"; N
20 IF N=0 THEN 80
30 IF N<1 OR N>10 THEN 10
40 IF N MOD 2=0 THEN 70
50 PRINT N;" is odd"
60 GOTO 80
70 PRINT N;" is even"
80 END
```

In line 20, if the value in the variable *N* (input in line 10) is 0, the program branches to line 80, which ends the program. In line 30, if the value in *N* is less than 1 or greater than 10 then the program branches to line 10 to reenter a number. In line 40, if the modulus of

the value in *N* and 2 is 0 then the number is even and the program branches to line 70 otherwise the program continues with line 50 before branching to line 80 skipping line 70.

5.3 Character Sets

Text in other languages can be displayed by using different character sets. To select a character set, poke one of the following values into location 29538:

0	ANSI (default)
161	Greek
162	Turkish
177	Hebrew
178	Arabic
186	Baltic
204	Russian
238	East Europe
255	OEM

To make sure the character set was successfully changed, compare the value returned by PEEK(29538) to the character set specified in the **POKE** statement.

On startup, the character set is set to ANSI (0).

The following example program displays the Russian word for hello:

```
10 POKE 29538,204
20 IF PEEK(29538)<>204 THEN PRINT "204 Not Available": END
30 PRINT
CHR$(239);CHR$(240);CHR$(232);CHR$(226);CHR$(229);CHR$(242)
40 POKE 29538,0
50 END
```

Line 10 selects the Russian character set. Line 20 checks if the character set was successfully changed. Line 30 prints the Russian word for hello. Line 40 selects the default ANSI character set.

To select a character set on the printer, poke one of the above values into location 46966 (Note: the printer must be opened before the character set can be changed).

5.4 Data

Information can be added to a program using **DATA** statements. A **DATA** statement contains numeric or string values separated by commas that are read using the **READ** statement. The **READ** statement reads the values in the **DATA** statements into variables. When all of the data in the **DATA** statements has been read by the **READ** statement, a subsequent **READ** statement will cause an "Out of data" error. To read the data again, use the **RESTORE** statement. The **RESTORE** statement resets the location where the **READ** statement begins reading data.

The following example program uses data statements, which contain the months of the year and the number of days in each month (Note: February has 29 days in a leap year). The program reads the data and prints out the list.

```
10 DATA "January",31,"February",28,"March",31
20 DATA "April",30,"May",31,"June",30
30 DATA "July",31,"August",31,"September",30
```

```
40 DATA "October",31,"November",30,"December",31
50 FOR I=1 TO 12
60 READ M$,D
70 PRINT M$;" has";D;" days"
80 NEXT I
90 END
```

The data values are contained in lines 10 through 40. The FOR loop in line 50 specifies the number of times to read data (twelve (12) in this case). Line 60 reads two (2) data values placing the first value, a string, in the string variable *M$* and the second value, a number, in the numeric variable *D*. Line 70 prints the values. Line 80 increments the loop variable *I* and repeats the loop if it has not reached the end value specified in the **FOR** statement in line 50 or ends the loop and continues with the next statement if it has. Line 90 ends the program.

Several methods can be used to determine when all the data has been read. The number of data values may be specified in the program as in the above example. A terminal value can be used in the **DATA** statements such as -1 for numeric values or an empty string ("") for string values. Also, the number of data elements can be specified in the **DATA** statement.

5.5 Data Types

Data can be of two types, numeric or string. A numeric data type represents a number in the range 1.7E-307 to 1.7E+308 with 15 digits of precision. A string represents a set of characters with a maximum length of 65,535.

Numbers are specified in decimal format and can optionally contain an exponent. A leading + or - sign can be included (+ is optional for positive numbers). Exponents can also be positive or negative.

Examples of number:
10
121.654
1.765E2
0.4
-48.5

Numbers can also be specified in hexadecimal, octal and binary formats. These numbers are whole numbers (integers) in the range 0 to 4,294,967,295. Hexadecimal numbers begin with &H and may contain up to eight hexadecimal digits (0-9,A-F). Octal numbers begin with &O and may contain up to eleven octal digits (0-7). Binary numbers begin with &B and may contain up to thirty-two binary digits (0 or 1).

Examples:
&H1F (31)
&O37 (31)
&B11111 (31)

Strings are specified using double quotes. The string "" is an empty string (contains no characters).

Examples of strings:
“Hello"
"ABC"
"100"

"a"
""
"""""

5.6 Date and Time

To get the current date and time use the **NOW** function. The **NOW** function returns a number that represents the date and time when the function was called. This number is used with other functions that provide date and time information.

To change the time value, use the **ADJUST** function. The **ADJUST** function can adjust the time by any number of seconds. Use a positive number to adjust the time forward or a negative number to adjust the time backwards.

To create a time value, use the **MKTIME** function. Specify the year, month and day and optionally the hour, minute and second and the **MKTIME** function returns a time value representing this date and time.

To get the date as "MM-DD-YYYY" or the time as "HH:MM:SS", use the **DATE$** or **TIME$** functions.

To get the year, month or day from the time value use the **YEAR**, **MONTH**, or **DAY** functions and to get the hour, minute and second use the **HOUR**, **MINUTE**, or **SECOND** functions.

5.7 Debugging

Removing problems in a program is called debugging. There are several statements to help in debugging. Use the **TRACE** statement to print the lines of a program as it executes. To stop at a specific line in a program, edit the line where the program should stop and insert a **STOP** statement. The **STOP** statement causes the program to exit but all of the program's information is kept. To continue the program, use the **CONT** statement. The **CONT** statement restarts the program where it was stopped. If any changes are made to the program, the program will have to be restarted using the **RUN** statement.

More advanced debugging statements allow the program to be executed one statement at a time, which makes program execution easier to follow than with the **TRACE** statement. Use the **STEP** statement to execute the next statement. After the statement is executed the line containing the next statement to be executed is printed with the statement in brackets. Breakpoints can be set in the program using the **BREAK** statement so that when execution reaches the line containing the breakpoint, the program will stop as if a **STOP** statement were executed. To continue from the breakpoint use the **CONT** or **STEP** statements. To clear a breakpoint, use the **NOBREAK** statement. The **ASSERT** statement can be used to check conditions in a program if the condition is not true the program will stop. In order for breakpoints and assertions to be active use the **DEBUG** statement. Debug mode can be turned on or off and is useful to run the program without stopping at breakpoints or assertions.

5.8 Errors

Errors that occur in a program will stop execution of the current program if not handled. Use the **ON ERROR** statement to trap errors. When an error occurs in a program execution is transferred to the line specified in the **ON ERROR** statement. The number of the error that occurred can be obtained using the **ERN** function or the line where the

error occurred using the **ERL** function. Use the **ERR$** function to get a description of the error. To return to the program from the error handler use the **RESUME** statement. The **RESUME** statement can retry the statement that caused the error, resume execution after the statement that caused the error or resume execution to a specific line. The **ON ERROR** statement can be used to disable error trapping by specifying 0 as the line number.

The following example program sets an error handler before writing a file:

```
10 ON ERROR GOTO 80
20 OPEN "TEST.DAT" FOR OUTPUT AS #1
30 FOR N=1 TO 10
40 PRINT #1,N
50 NEXT N
60 CLOSE #1
70 END
80 PRINT "Error: ";ERR$(ERN);" in line";ERL: RESUME NEXT
```

Line 10 sets the error handler to line 80. Lines 20 through 60 open a file, write the numbers 1 through 10 to the file and then close the file. If any errors occur in these lines, execution is transferred to line 80, which prints a description of the error and the line where the error occurred before resuming the program with the next statement after the statement that caused the error. If no errors occur, the programs ends in line 70.

5.9 Expressions

An expression is a sequence of operators and operands that are used to compute a value. The result of an expression is either a numeric or string value.

Expressions are evaluated according to the precedence and grouping of the operators in the expression.

For example, in the expression 4+2*3, multiplication (*) has a higher precedence than addition (+) and is evaluated first. So the result of this expression is 10. Grouping uses parentheses to specify how operators are evaluated. For example, in the expression (4+2)*3, (4+2) is evaluated first because of grouping. So the result of this expression is 18.

An operand can be a numeric or string literal, constant, variable, function, user defined function or another expression.

A type mismatch error occurs if the wrong data type is used with an operator, statement or function.

5.10 Files

A file is the basic unit of storage and is kept on a storage medium such as disk or tape. Files are organized in groups called volumes.

To access the contents of a file, the file must be open. To open a file, use the **OPEN** statement. A file can be opened for sequential input or output, or for random access. When a file is opened for output, a new file is created or if the file already exists, it is truncated. Data written to the file is appended to the end of the file. A file opened for input must already exist. Data is read from the file starting at the beginning of the file. To write data to a file, use the **PRINT**, **PRINT USING**, or **WRITE** statements. To read data from a file, use the **INPUT** or **LINE INPUT** statements or the **INPUT$** function. To check

when the end of file has been reached, use the **EOF** function. When finished using a file, it should be closed. To close a file, use the **CLOSE** statement.

Data is read from and written to the file's buffer. The file buffer is used to temporarily hold the data to minimize disk access and therefore increases the speed of file input and output. When writing to a file and the buffer becomes full, the buffer is written to disk and emptied. When reading from a file and the buffer is empty, data is read from disk into the buffer. The buffer size may be specified in the **OPEN** statement.

When a file is opened for random access, data can be both read from and written to the file in the form of records. The size of the record is specified in the **OPEN** statement as the buffer size. A record can have one (1) or more fields. To define a record's fields within the file, use the **FIELD** statement. The record (buffer) size of the file must be at least the size of all the fields defined in the **FIELD** statement. A field is a string of a specified size that contains either string or numeric data. A numeric field must always be defined with a size of eight (8). To write to a field, use the **LSET** or **RSET** statements then use the **PUT** statement to write the record to disk. To read from a field, use the **GET** statement to read a record from disk and then use the name of the field to access the field's data. Use the **MKN$** and **CVN** functions to convert numeric data to and from a binary string for use in fields.

The following example program demonstrates sequential input and output. The program writes numbers to a file, then reads the numbers from the file and prints them.

```
10 OPEN "TEST.DAT" FOR OUTPUT AS #1
20 FOR I=1 TO 10
30 PRINT #1,I
40 NEXT I
50 CLOSE #1
60 OPEN "TEST.DAT" FOR INPUT AS #1
70 INPUT #1,I
80 PRINT I
90 IF NOT EOF(1) THEN 70
100 CLOSE #1
110 END
```

Line 10 opens the file "TEST.DAT" in the default volume for output as file number 1. Lines 20 and 40 contain a FOR loop which counts from 1 to 10 using the numeric variable *I*. Line 30 prints the number in the variable *I* to the file (file number 1). Line 50 closes the file. Line 60 opens the file for input. Line 70 inputs a single numeric value from the file and assigns it to the numeric variable *I*. Line 80 prints the number in the variable *I*. Line 90 checks if the end of file has been reached and if not goes to line 70 to input another value. If the end of file has been reached program execution continues with line 100. Line 100 closes the file. Line 110 ends the program.

The following example demonstrates a random mode file. The program writes information to a file, reads the file and formats the information in a table.

```
10 DATA "A-Sciences Corp.",234.59
20 DATA "Data Enterprises",1088.21
30 DATA "NBA Systems",298.44
40 DATA "TRR Inc.",541.65
50 DATA "",0
60 OPEN "INVOICES.DAT" FOR RANDOM AS #1 LEN=28
70 FIELD #1,20 AS AA$,8 AS BB$
80 R=1
```

```
90 READ A$,B
100 IF A$="" THEN 140
110 LSET AA$=A$: LSET BB$=MKN$(B)
120 PUT #1,R
130 R=R+1: GOTO 90
140 PRINT "Account" TAB(21) "Balance"
150 PRINT STRING$("-",20);" ";STRING$("-",10)
160 R=1
170 GET #1,R: IF EOF(1) THEN 200
180 PRINT USING "\                   \ $$####,.##";AA$,CVN(BB$)
190 R=R+1: GOTO 170
200 CLOSE #1
210 END
```

Lines 10 through 50 contain the data to be written to the file. Line 60 opens the file "INVOICES.DAT" as a random mode file with a record length of 28. Line 70 divides the file buffer into fields. The first field (*AA$*) will contain the account name and up to twenty (20) characters. The second field (*BB$*) will contain the account balance and is 8 characters in length (all numeric fields must be 8 characters in length). The record length of the file in the **OPEN** statement must be at least the size of all the field lengths combined. Line 80 sets the numeric variable *R* to 1, which is used for the record number (record numbers begin at 1). Line 90 reads the account name into the string variable *A$* and the balance into the numeric variable *B*. Line 100 checks if all the data has been read using the special marker in line 50, an empty string. Line 110 copies the data in the variables to the file buffer. The **LSET** statement left justifies the string in the string variable *A$* into the field variable AA$. Numeric values have to be converted to binary strings before copying to a field variable because all field variables are strings (of fixed size). A field that is to contain a numeric value must be 8 characters in length. To convert a numeric value to a binary string, use the **MKN$** function. Line 120 writes the file buffer to the record number contained in the numeric variable *R*. Line 130 increments the record number in the variable *R* and goes to line 90 to read the next account information.

Line 100 branches to line 140 if the end of data marker is read. Lines 140 and 150 print column headers for the account information. Line 160 sets the record number to 1, the record number of the first record in the file. Line 170 gets the record from the file into the field variables (*AA$* and *BB$*). If the end of file has been reached the program branches to line 200. Line 180 prints the formatted account information. Line 190 increments the record number and goes to line 170 to get the next record. Line 200 closes the file. Line 210 ends the program.

To list the files in a volume, use the **DIR** statement. To delete a file, use the **KILL** statement.

To copy a file, use the **COPY** statement. To move a file, use the **MOVE** statement. To rename a file, use the **RENAME** statement.

5.11 Functions

Functions can be used where an expression is allowed. To call a function, specify the function's name and any arguments the function requires. Functions always return a value unless an error occurs. Function names are case insensitive.

Function names that end with a dollar sign ($) return a string and functions that do not end with a dollar sign return a number. If a function does not accept any arguments do not include the opening and closing parentheses in the function call.

5.12 Graphics

The graphics screen size varies due to the display resolution and the current font selected. Graphics programs should determine the screen size and adjust the display of graphics output accordingly. To get the screen width and height in pixels, use the **PSCRW** and **PSCRH** functions. The pixels are numbered 0 to the maximum size minus 1. Therefore, if the **PSCRW** function returned 800 as the width of the graphics screen in pixels, the pixels are numbered 0 to 799.

Before graphics can be drawn, graphics must be turned on and initialized. To begin drawing graphics, use the **GRAPH** statement. Text and graphics can be mixed on the same screen.

To set a pixel with a specific color, use the **PSET** or **PRESET** statements. To clear the graphics screen, use the **PCLS** statement. To set the graphics foreground and background colors, use the **PCOLOR** statement. To draw lines, rectangles, and circles, use the **LINE** and **CIRCLE** statements. To print text anywhere on the graphics screen, use the **PPRINT** statement.

The following example program displays a circle, 100 pixels in diameter, in the center of the screen:

```
10 GRAPH ON
20 PCLS
30 CIRCLE (PSCRW/2,PSCRH/2),50
40 WAIT 2000
50 GRAPH OFF
60 END
```

The program first turns graphics on (line 10) then clears the graphics screen (line 20), draws the circle (line 30), waits for two (2) seconds (line 40), turns graphics off (line 50) and ends (line 60).

5.13 Keyboard

The keyboard is used for user input. Use the **INPUT** statement to prompt for input and to read the data into variables. Use the **INPUT$** function to read a specific number of characters. Use the **LINE INPUT** statement to read a single line of data.

5.14 Limits

Numeric range:	1.7E-307 to 1.7E+308
Maximum string length:	65535 characters
Maximum variable name length:	31 characters
Maximum user defined function name length:	31 characters
Maximum label name length:	31 characters
Maximum constant name length:	31 characters
Maximum array dimensions:	10
Maximum array dimensions if unspecified:	3

Maximum array size per dimension:	65536
Maximum array size all dimensions:	16777216
Minimum user defined function arguments:	0
Maximum user defined function arguments:	10
Maximum number of lines in a program:	65535
Maximum number of variables in a program:	Available memory
Maximum number of user defined functions in a program:	Available memory
Maximum number of labels in a program:	1 per line
Maximum number of constants in a program:	Available memory
Maximum number of nested FOR loops:	Available memory
Maximum number of GOSUB calls:	Available memory
Maximum volume name length:	15 characters
Maximum number of volumes:	Available memory and disk space
Maximum size of sequential or random access file:	2GB
Maximum number of open files:	255
Maximum file buffer length:	32768
Maximum number of fields in a random access file buffer:	128
Maximum size of a log file:	2GB
Maximum number of files in a volume:	Available disk space
Maximum NBASIC line length:	255

5.15 Logging

Logging is useful in recording the output of a program or an NBASIC session to a file. Everything that is output to the screen or input from the keyboard is written to the log file.

To open a log file, use the **LOG** statement. Once the log file is open, recording to the log file begins. You can stop recording using `LOG OFF`, to begin recording again use `LOG ON`. To close the log file use `LOG STOP`.

5.16 Loops

Repeating one or more of statements is called a loop. One way to repeat a group of statements is to use the **GOTO** statement to branch to the beginning of the statement group. An **IF** statement can be used to determine when the loop terminates and then branch to a line outside the loop.

To repeat a group of statements a specific number of times, use the **FOR** and **NEXT** statements. A FOR loop initializes a numeric variable to a beginning value, and either increments or decrements the variable until an ending value is reached (note: a FOR loop is always executed at least once).

The following example program waits for a key to be pressed before printing the numbers 1 through 10:

```
10 PRINT "Press a key to continue"
20 LET K$=INKEY$: IF K$="" THEN 20
30 FOR N=1 TO 10
40 PRINT N
50 NEXT N
60 END
```

Line 20 contains a loop then checks for a key press and if no key has been pressed loops back to line 20 to check again. The loop is repeated until a key is pressed.

Lines 30 through 50 contain a FOR loop. Line 30 begins the loop, setting the numeric variable *N* to 1. Line 40 prints the value of the variable *N*. Line 50 increments the variable *N* by 1 and tests if it is above the end value specified in line 30. If the value in *N* is not above the end value of the loop, the **NEXT** statement repeats the statements after the **FOR** statement in line 30 otherwise execution continues with the statement after the **NEXT** statement.

5.17 Printing

Programs can print text to a printer.

In order to send output to a printer, the printer must be opened. To open a printer, use the **OPEN PRINTER** statement. The **OPEN PRINTER** statement opens the printer set by the **PRINTER** statement or if no printer has been set, the current printer specified in the Print Setup or Print dialogs. The **PRINTER?** statement can be used to set the printer by allowing the printer to be selected from a list of installed printers. To send output to the printer, use the **LPRINT** or **LPRINT USING** statements. Output can also be sent to the printer using other print statements by using the file number -2. To get the current print position (column), use the **LPOS** function. To close the printer and finish printing, use the **CLOSE PRINTER** statement.

The following example program prints a simple message on the printer:

```
10 PRINT "Printing version information..."
20 PRINTER?: REM select a printer
30 OPEN PRINTER
40 LPRINT "Version information: ";VER$;
50 LPRINT ", Release ";STR$(PEEK(36))
60 CLOSE PRINTER
70 END
```

Line 20 displays a dialog to select the printer to use as the current printer. Line 30 opens the printer. Lines 40 and 50 send output to the printer. Line 60 closes the printer, finishes printing and ends the print job.

5.18 Programs

NBASIC programs consist of lines identified by line numbers. Each program line may contain one (1) or more statements. Line numbers start at 1 and may continue up to 65535. Typically, line numbers in a program begin with 10 with each succeeding line incremented by 10 (20,30,40...). This convention makes it easy to add lines to a program without renumbering.

A program line is referenced by its line number (e.g. `GOTO 50`, `EDIT 80`) and may also include a label. A label is a unique name with an asterisk (*) preceding it and can be used instead of a line number in the program.

When a program is run, execution of the program begins with the lowest numbered line and proceeds sequentially to higher numbered lines or to lines specified by the program using a branch statement (**IF**, **GOTO**, etc).

To delete the current program from memory, use the **NEW** statement. Note that once the program has been deleted it cannot be recovered unless it has been saved to disk.

To save a program to disk, use the **SAVE** statement. A program can be saved as a binary or ASCII file. An ASCII file contains only the source code of the program while a binary file contains the compiled version of the program. To load a program from disk use the **LOAD** statement.

To renumber a program, use the **RENUM** statement. To view a program, use the **LIST** statement. To edit a line, use the **EDIT** statement. To delete a line or lines, use the **DEL** statement or type the line number at the ready prompt and press enter.

A running program can be interrupted using the BREAK key or paused using the PAUSE key.

5.19 Screen

The standard NBASIC text screen is 80 columns wide numbered 0 through 79 and 25 rows high numbered 0 through 24. NBASIC can display sixteen colors numbered 0 through 15.

The cursor can be turned on or off using the **CURSOR** statement. Input statements always display the cursor. The position of the cursor also indicates where output will be printed. To set the cursor or print location, use the **LOCATE** statement. To clear the entire screen, use the **CLS** statement. To get the current position of the cursor, use the **ROW** and **COLUMN** functions.

To change the foreground and background colors, use the **COLOR** statement.

To print to the screen, use the **PRINT** or **PRINT USING** statements.

Output is wrapped to the next line if it extends beyond the right edge of the screen and is scrolled if it extends beyond the last line.

NBASIC Colors	
Black	0
Dark Red	1
Dark Green	2
Dark Yellow	3
Dark Blue	4
Dark Magenta	5
Dark Cyan	6
Light Gray	7
Medium Gray	8
Red	9

Green	10
Yellow	11
Blue	12
Magenta	13
Cyan	14
White	15

5.20 Sound

Programs may use sound in several ways. The **BEEP** statement plays the standard beep on the computer's speakers. The **SOUND** statement can play a sound with a frequency in the range of 37Hz to 32,767Hz for the specified number of milliseconds.

5.21 Statements

Statements can be entered in immediate mode or as part of a program. To enter a statement type the statement's name and any parameters the statement requires. Each statement has syntax rules specifying how the statement can be entered and what parameters are allowed. If a statement is entered incorrectly a syntax error occurs indicating that the statement was entered incorrectly. The statement must be corrected before it can be executed. Statement names are case insensitive. Multiple statements can be entered in a single line if a colon separates them.

5.22 Subroutines

Subroutines are groups of statements that perform a useful task that needs to be repeated. Subroutines are called using the **GOSUB** statement. To return from a subroutine use the **RETURN** statement.

For example, the subroutine in the following program is called using `GOSUB 100` in line 20. The subroutine prints a string and then returns to the calling program by `RETURN` in line 110.

```
10 FOR I=1 to 10
20 GOSUB 100
30 NEXT I
40 END
100 PRINT STR$(I)+ ":"+STRING$("*",I)
110 RETURN
```

A subroutine can contain any number of lines within the limits of the program and must contain at least one **RETURN** statement.

Subroutines can use the variables already defined in a program. There is no parameter passing mechanism with this type of subroutine.

5.23 Timers

NBASIC provides several types of timers that can be used in your programs.

The most common use for a timer is to pause execution of a program for a specified amount of time. The **WAIT** statement can be used for this purpose. The **WAIT** statement pauses until the specified amount of time elapses. For example, the statement `WAIT 2000` pauses until 2000 milliseconds (2 seconds) have elapsed before continuing.

Another common use for a timer is to time an event. The **TIMER** function returns the current value of the timer in milliseconds. The following example program determines the time (milliseconds) required to count from 1 to 100:

```
10 S=TIMER
20 FOR I=1 TO 100: NEXT I
30 E=TIMER
40 PRINT "TIME:";E-S;" MILLISECONDS"
50 END
```

The program first gets the starting timer value, counts from 1 to 100 in a loop, and then gets the ending timer value. The difference in the start and end timer values is displayed.

The timer is set to 0 when NBASIC starts. To reset the timer use the **TIMER** statement. For example, the statement `TIMER=0` resets the timer to 0.

A timer can also interrupt a program at a specified interval, transferring control to a timer handler. The **ON TIMER** statement is used to setup a timer. Specify the timer interval in milliseconds and the line number or label of the timer handler. In the timer handler, use the **RESUME** statement to return control to the program.

5.24 User defined functions

You can define your own functions in NBASIC using the **DEF FN** statement. A user-defined function must be defined and the program line containing the definition must be executed before it can be used. User-defined functions must be given a name. The name must start with a letter and can be up to thirty-one (31) characters in length and is case insensitive. User-defined functions can accept up to ten (10) arguments and return a single numeric result.

The following user-defined function defines a function named *DOUBLE* accepting a single numeric argument named *X* which will compute the expression X*2:

```
10 DEF FNDOUBLE(X)=X*2
```

The arguments in a function definition must be valid variable names. In this example, *X* is used as the argument name. If there is a variable *X* defined elsewhere in the program it cannot be accessed within the function definition. *X* is used in the function definition to refer to the argument of the function not the global variable *X*. Other variable names can be used in the function definition as long as there is no argument defined in the function with the same name.

To call the function, use the **FN** function. The following example calls the function *DOUBLE*:

```
20 LET A=FNDOUBLE(5)
```

The user-defined function *DOUBLE* is called with an argument of 5. NBASIC looks up the function name and substitutes 5 for the argument *X*. The result of X*2, in this case 10 (5*2), is returned as the result and assigned to the variable *A*. Any numeric expression can be passed as the argument. If a string value is passed as the argument, a Type mismatch error occurs.

A user-defined function can be defined without any parameters. For example,

```
10 DEF FNHALFPI=1.5707963267949
```

defines a function that returns a constant value (Pi/2).

Functions can be defined with multiple arguments, for example,

```
10 DEF FNAVG(X,Y)=(X+Y)/2
```

defines a function that accepts two numeric arguments *X* and *Y*. The function computes the average of the two numbers.

User defined functions can also accept string parameters. For example,

```
10 DEF FNCNTRSTR(S$,W)=(W-LEN(S$))/2
```

defines a function that accepts a string argument *S$* and a numeric argument *W*.

User defined functions can include other user-defined functions in the definition provided that they have already been defined. User defined functions cannot be called recursively, which means they cannot reference themselves in their own definition.

5.25 Variables

Variables contain the data a program is using and can be either numeric or string.

Variable names can be up to thirty-one (31) characters, may contain letters or digits but must begin with a letter and are case insensitive. Variables names ending with a dollar sign ($) are string variables and variables that do not end in a dollar sign are numeric. Variable names cannot begin with the names of statements or built-in functions.

Variables are created when they are first used. Numeric variables are initialized to zero (0) while string variables are initialized to an empty string.

The single letter numeric variables named A-Z are statically allocated by the system and are always available and faster then normally allocated variables.

To delete all variables, use the **INIT** statement.

6
How To

6.1 Access The Context Menu

To access the context menu, close any dialogs that may be open in NBASIC. Right click the mouse anywhere within the NBASIC window client area. The context menu will be displayed where you clicked the mouse. Use the mouse or keyboard to select options from the menu.

6.2 Create A Volume

To create a new volume, you map a Windows folder to a NBASIC volume name. Right-click the mouse anywhere within the NBASIC client window to display the context menu (in command mode). Select Volumes... in the menu to display the Volumes dialog. Click on the Add.. button to display the Add dialog. Enter the name of the volume up to fifteen characters (only letters and numbers are allowed). Enter the name of the Windows folder to map to the volume or click Browse.. to browse for a Windows folder. If you want the volume to be the default volume select the Default check box. If you want the volume to be read-only select the Read Only check box. Read only volumes do not allow files to be created, modified, renamed or deleted. Click on OK to create the new volume. The new volume will be displayed in the list. Click on Close to close the Volumes dialog box.

6.3 Get Help

There are a number of ways to get help while using NBASIC.

You can press the F1 key at anytime to display a list of help topics from which to choose from.

Or you can right-click anywhere within the NBASIC client window to display the context menu and select Help Topics to display the help table of contents or index.

All of the dialogs used in NBASIC have context help by clicking on the question mark (?) at the top right corner of the dialog, which can be used to display help about each control in the dialog. You can also get an overview of the entire dialog by clicking the Help button.

You can also use the **HELP** statement. Type `HELP` and press the ENTER key to get a list of help topics or type `HELP` followed by a topic you wish to get help on in quotes. For example, typing `HELP "cls"` and pressing the ENTER key displays all help topics that match "cls". Select the help topic you want help on from the list.

6.4 Load A Program

To load an existing program, use the **LOAD** statement.

In immediate mode, type `LOAD` followed by the name of the program to load enclosed in double quotes and press the ENTER key. If the program is located in a volume that is not the default volume, include the volume name as part of the file name.

For example, **LOAD** "HELLO.BAS" loads the program HELLO.BAS from the default volume. **LOAD** "GAMES:CARD.BAS" loads the program CARD.BAS from the volume GAMES.

To run the program immediately use **,R** after the file name (e.g. `LOAD "HELLO.BAS",R`).

6.5 Print A Program

To print the current program, right click the mouse in the NBASIC client window to display the context menu. Select Print... from the context menu. Select the printer and print options and click OK.

To print specific lines in the current program, use the **LLIST** statement.

6.6 Save A Program

To save the current program, use the **SAVE** statement.

In immediate mode, type SAVE followed by the name of the program to load enclosed in double quotes and press the ENTER key. If the program is to be saved on a volume that is not the default volume, include the volume name as part of the file name. The **SAVE** statement will overwrite an existing file with the same name.

For example, `SAVE "HELLO.BAS"` saves the program HELLO.BAS on the default volume. `SAVE "GAMES:CARD.BAS"` saves the program CARD.BAS on the volume GAMES.

To save the file as ASCII use **,A** after the file name (e.g. `SAVE "HELLO.BAS",A`).

Part II

NBASIC Reference

Part II provides a reference for the statements and functions available in NBASIC. It includes a list of error codes that may be encountered in using NBASIC. Part II also contains a reference for control and editing keys used in NBASIC.

7
Statements

7.1 **$COLOR Statement** (meta-command)

Restores the default foreground and background colors when a program ends.

$COLOR

Remarks

If the **$COLOR** statement is placed in a program, when the program ends (or stops), the default foreground and background colors are restored.

The **$COLOR** statement can be specified anywhere in the program.

Mode

Program only

7.2 **$PRINT Statement**

Prints the current program.

$PRINT

Remarks

The **$PRINT** statement performs the same function as selecting Print... from the context menu.

Mode

Immediate only

See Also

LLIST

7.3 **$XREF Statement**

Cross-references the current program.

$XREF

Remarks

The **$XREF** statement lists each line that is referenced in the program and the lines containing the reference.

Mode

Immediate only

7.4 * Statement

Defines a label.

***label**

Parameters

label
Name of label.

Remarks

Labels can be used anywhere a line number is used.

The label definition must be the first statement in the line and only one label definition is permitted. Label names can be up to 31 characters and must begin with a letter.

Examples

```
10 *BEGIN: CLS
```

Mode

Immediate, Program

See Also

GOTO, **GOSUB**

7.5 ABOUT Statement

Displays program information, version number, and copyright.

ABOUT

Mode

Immediate, Program

See Also

VER

7.6 APPEND Statement (standard version only)

Appends a file to another file.

APPEND *filespec1* **TO** *filespec2*

- or -

A. *filespec1* **TO** *filespec2* (Version 1.5 or later)

Parameters

filespec1
Name of existing file.

filespec2
Name of new or existing file.

Remarks

The **APPEND** statement can be used to append an existing file to another file.

The *filespec* and *filespec2* parameters may contain the wildcard characters * and ?. (Version 1.5 or later)

Examples

```
APPEND "TEST.DAT" TO "CONFIG.DAT"
APPEND "PROGRAMS:TEST.DAT" TO "DATA:TEST.DAT"
APPEND "*.LOG" TO "HISTORY.TXT"
APPEND "*.TXT" TO "*.DOC"
```

Mode

Immediate, Program

See Also

COPY, **MOVE**, **RENAME**

7.7 APPEND Statement, editing (standard version only)

Appends a line in the current program to another line.

APPEND *line1* **TO** *line2*[**,**{**E** | **D** | **ED**}]

- or -

A. *line1* **TO** *line2*[**,**{**E** | **D** | **ED**}] (Version 1.5 or later)

Parameters

line1
Line number of line to append.

line2
Line number of destination line.

Remarks

The **APPEND** statement appends an existing line in the current program to another existing line.

Use **,E** to edit the destination line and/or **,D** to delete the line being appended.

Examples

```
APPEND 80 TO 70
APPEND 150 TO 140,E
APPEND 150 TO 140,ED
APPEND 205 TO 200,D
```

Mode

Immediate only

See Also

COPY, **MOVE**, **SWAP**

7.8 ARC Statement (standard version only)

Draws an arc.

ARC (*x***,***y***),***radius*[**,**[*color*][**,**[*start*][**,**[*end*][**,**[*aspect*][**,F**]]]]]

Parameters

x
X coordinate of center of circle.

y
Y coordinate of center of circle.

radius
Radius of arc.

color
Color of arc.

start
Starting angle of arc in radians.

end
Ending angle of arc in radians.

aspect
Ratio of length of y-axis to length of x-axis used to draw ellipses.

Remarks

The **ARC** statement can be used to draw circles, ellipses, and arcs. To draw a circle *aspect* must be 1 or omitted. To draw ellipses *aspect* can be less than 1 for ellipses stretched on the x-axis or greater than 1 for ellipses stretched on the y-axis. To draw arcs specify the *start* and *end* angles.

Use **,F** to fill in a circle.

If *color* is omitted the default is the current graphics foreground color. If *start* is omitted the default is 0. If *end* is omitted the default is 0. if *aspect* is omitted the default is 1.

Graphics must be turned on to draw circles.

The **ARC** statement is equivalent to the **CIRCLE** statement and is provided for compatibility.

Examples

```
ARC (100,100),50
ARC (200,200),10,5,,,F
ARC (150,150),100,,,2
```

```
ARC (150,150),100,,,,.5
ARC (50,50),25,,.785,3.142
```

Mode

Program only

See Also

CIRCLE

7.9 ASAVE Statement (Version 1.4 or later)

Saves a program.

ASAVE *filespec*

Parameters

filespec
File name of program to save.

Remarks

The **ASAVE** statement can be used to save the current program. The program is saved in ASCII format. By convention, programs in ASCII format have an extension of "bas".

The default extension for the **ASAVE** statement if not specified in the *filespec* is "nba".

Examples

```
ASAVE "ANALYZE"
ASAVE "PROGRAMS:COUNT.BAS"
```

Mode

Immediate, Program

See Also

BSAVE, **LOAD**, **SAVE**

7.10 ASSERT Statement (standard version only)

Conditionally stops the current program.

ASSERT *condition*[**,***message*]

Parameters

condition
Expression that evaluates to true or false.

message
String to display if *condition* is false.

Remarks

The **ASSERT** statement can be used to verify conditions in a program being tested. If *condition* is false, the program is stopped and *message* is displayed.

Debugging must be turned on for asserts to stop execution.

Examples

```
ASSERT ITEMS>100
ASSERT INDEX<>0,"INDEX IS ZERO!"
```

Mode

Immediate, Program

See Also

DEBUG

7.11 AT Statement

Moves the cursor to a specified position on the screen.

AT *row*,*column*

Parameters

row
Row where cursor is to be moved.

column
Column where cursor is to be moved.

Remarks

The screen has 25 rows numbered 0 through 24 and 80 columns numbered 0 through 79.

The **AT** statement is equivalent to the **LOCATE** statement and is provided for compatibility.

Examples

```
AT 10,0
```

Mode

Immediate, Program

See Also

LOCATE

7.12 ATTACH Statement (Version 1.5 or later)

Attaches screen output to a file.

ATTACH {*#file* | **LPRINT**}

Parameters

file
File number of open file or printer (-2) to attach.

Remarks

The **ATTACH** statement causes screen output to be copied to a file or printer.

Only one file can be attached at a time.

Examples

```
ATTACH #1
ATTACH LPRINT
```

Mode

Immediate, Program

See Also

DETACH, **OPEN**

7.13 ATTRIB Statement

Displays or changes the read-only attribute of a file.

ATTRIB *filespec*

- or -

AT. *filespec* (Version 1.5 or later)

- or -

ATTRIB {**+** | **-**}**R** *filespec*

- or -

AT. {**+** | **-**}**R** *filespec* (Version 1.5 or later)

Parameters

filespec
File name of file whose read-only attribute will be displayed or changed.

Remarks

The **ATTRIB** statement can be used to display the read-only attribute of a file. If **+R** or **-R** is specified the read-only attribute is either added or removed.

The default extension for the **ATTRIB** statement if not specified in the *filespec* is "nba".

The *filespec* parameters may contain the wildcard characters * and ?. (Version 1.5 or later)

Examples

```
ATTRIB "TEST.DAT"
ATTRIB +R "TEST.DAT"
ATTRIB -R "TEST.DAT"
ATTRIB "*.DAT"
ATTRIB +R "*.TXT"
```

Mode

Immediate only (first form), Immediate or Program (second form)

See Also

ATTRIB$

7.14 AUTO Statement (standard version only)

Turns automatic line numbering on or off.

AUTO ON[**,***increment*]

- or -

AUTO OFF

Parameters

increment
Increment for each new line number.

Remarks

The **AUTO** statement can be used to automatically generate line numbers when entering a program. After each line is entered, the line number is incremented by *increment* and is automatically entered as part of the next line. If a line already exists with this line number, a new line number is not automatically generated.

If *increment* is omitted the default is 10.

Examples

```
AUTO ON
AUTO ON,10
AUTO OFF
```

Mode

Immediate only

7.15 BACKUP Statement

Copies files in a volume to another volume.

BACKUP *volume1* **TO** *volume2*

- or -

B. *volume1* **TO** *volume2* (Version 1.5 or later)

Parameters

volume1
Volume to copy files from.

volume2
Volume to copy files to.

Remarks

The **BACKUP** statement can be used to backup all the files in a volume to another volume.

Example

```
BACKUP "DATA" TO "ARCHIVE"
```

Mode

Immediate, Program

7.16 BEEP Statement

Generates a beep sound from the computer's speakers.

BEEP

Remarks

The sound generated depends on the host operating system's sound settings.

Mode

Immediate, Program

See Also

SOUND

7.17 BKOFF Statement (Version 1.4 or later)

Turns trapping of the BREAK key off.

BKOFF

Remarks

Pressing the BREAK key will stop the current program or statement. If trapping is turned off, the BREAK key is ignored.

Mode

Immediate, Program

See Also

BKON, **ON BREAK**

7.18 BKON Statement (Version 1.4 or later)

Turns trapping of the BREAK key on.

BKON

Remarks

Pressing the BREAK key will stop the current program or statement. If trapping is turned off, the BREAK key is ignored.

Mode

Immediate, Program

See Also

BKOFF, **ON BREAK**

7.19 BOX Statement (standard version only)

Draws a box.

BOX (*x1*,*y1***)-(***x2*,*y2***)**[,[*color*][**,F**]]

Parameters

x1
X coordinate of corner of box.

y1
Y coordinate of corner of box.

x2
X coordinate of opposite corner of box.

y2
Y coordinate of opposite corner of box.

color
Color of box.

Remarks

The **BOX** statement can be used to draw boxes. A box is a rectangle specified by the corner coordinates (*x1*,*y1*) and the opposite corner coordinates (*x2*,*y2*).

Use **,F** to fill in the box.

If *color* is omitted the default is the current graphics foreground color.

Graphics must be turned on to draw boxes.

Examples

```
BOX (10,10)-(100,100)
BOX (20,20)-(40,40),5
BOX (25,25)-(30,30),8,F
```

Mode

Program only

See Also

CHORD, **CIRCLE**, **DRAW**, **GRAPH**, **LINE**, **PAINT**, **PCOLOR**, **PIE**

7.20 BREAK Statement

Turns trapping of the BREAK key on or off.

BREAK {**ON** | **OFF**}

Remarks

Pressing the BREAK key will stop the current program or statement. If trapping is turned off, the BREAK key is ignored.

Examples

```
BREAK ON
BREAK OFF
```

Mode

Immediate, Program

See Also

BKOFF, **BKON**, **ON BREAK**

7.21 BREAK Statement, debugging (standard version only)

Lists or sets breakpoints.

BREAK

- or -

BREAK *line*[**,***line*]...

- or -

BREAK [*line*][**-**[*line*]][,[*line*][**-**[*line*]]]... (Version 1.2 or later)

Parameters

line
Line number or label where breakpoint is to be set.

Remarks

The **BREAK** statement can be used to set breakpoints in a program being tested. When a line that has a breakpoint set is executed, the program stops at that line and control is returned to immediate mode.

BREAK with no parameters lists all breakpoints.

The **CONT** statement can be used to continue execution after a breakpoint stops a program.

Debugging must be turned on for breakpoints to stop execution.

Examples

```
BREAK
BREAK 100,150
BREAK 200,*REDO
BREAK -10,20,50-100,900-
```

Mode

Program only (first form), Immediate or Program (second form)

See Also

DEBUG, **NOBREAK**

7.22 BSAVE Statement (Version 1.4 or later)

Saves a program.

BSAVE *filespec*

Parameters

filespec
File name of program to save.

Remarks

The **BSAVE** statement can be used to save the current program. The program is saved in NBASIC binary format. By convention, programs in binary format have an extension of "nba".

The default extension for the **BSAVE** statement if not specified in the *filespec* is "nba".

Examples

```
BSAVE "ANALYZE"
BSAVE "PROGRAMS:COUNT.NBA"
```

Mode

Immediate, Program

See Also

ASAVE, **LOAD**, **SAVE**

7.23 BYE Statement

Exits NBASIC.

BYE

Remarks

The **BYE** statement exits NBASIC and prompts to save the current program if modified.

The **BYE** statement is equivalent to the **EXIT** statement and is provided for compatibility.

Mode

Immediate only

See Also

EXIT

7.24 CALL CLEAR Statement (Version 1.4 or later)

Clears the screen.

CALL CLEAR

Remarks

The **CALL CLEAR** statement clears the screen using the current background color and sets the cursor position to 0,0.

The **CALL CLEAR** statement is equivalent to the **CLS** statement and is provided for compatibility.

Mode

Immediate, Program

See Also

CALL SCREEN, **CLS**

7.25 CALL SCREEN Statement (Version 1.4 or later)

Sets the screen display color.

CALL SCREEN(*color***)**

Parameters

color
Background text color.

Remarks

The **CALL SCREEN** statement sets the background text color, clears the screen and sets the cursor position to 0,0.

Mode

Immediate, Program

See Also

CALL CLEAR, **CALL VCHAR**

7.26 CALL VCHAR Statement (Version 1.4 or later)

Displays a character

CALL VCHAR(*row***,***column***,***characer*[**,***repeat*]**)**

Parameters

row
Starting row.

column
Starting column.

character
Character to display

repeat
Number of characters to display.

Remarks

The **CALL VCHAR** statement displays a character in one or more rows.

If *repeat* is omitted the default is 1.

Examples

```
CALL VCHAR(0,0,"*")
CALL VCHAR(0,0,"#",25)
```

Mode

Immediate, Program

See Also

CALL CLEAR, **CALL SCREEN**

7.27 CATALOG Statement (Version 1.1 or later)

Lists files on a volume.

CAT[**ALOG**]

Remarks

The **CATALOG** statement lists the files on a volume and information about each file.

Examples

```
CATALOG
CAT
```

Mode

Immediate only

See Also

DIR

7.28 CHAIN Statement

Transfers control from the current program to another program.

CHAIN *filespec*

- or -

CHAIN *filespec*,*line*

Parameters

filespec
File name of program to transfer control to.

line
Line number or label where execution is to start.

Remarks

The **CHAIN** statement transfers control to the specified program without initializing variables or closing open files.

The default extension for the **CHAIN** statement if not specified in the *filespec* is "nba".

Examples

```
CHAIN "ANALYZE"
CHAIN "CAPTURE.BAS",100
CHAIN "PROGRAMS:SORT.BAS",*START
```

Mode

Immediate, Program

See Also

RUN

7.29 CHKIO Statement

Checks for lines in the current program that contain file input/output statements.

CHKIO [*line*][-[*line*]]

- or -

CHKIO [*line*][**-**[*line*]][,[*line*][**-**[*line*]]]... (Version 1.2 or later)

Parameters

line
Line number of line to check.

Remarks

The **CHKIO** statement is useful to check programs downloaded from the Internet for statements that may delete or overwrite files.

If *line* is omitted all lines are checked.

Examples

```
CHKIO
CHKIO 100
CHKIO 10-
CHKIO 10-100
CHKIO -100
CHKIO -10,20,50-100,900-
```

Mode

Immediate only

7.30 CHKSW Statement (standard version only)

Checks for lines in the current program that contain non-shareware statements or functions.

CHKSW [*line*][**-**[*line*]]

- or -

CHKSW [*line*][**-**[*line*]][,[*line*][**-**[*line*]]]... (Version 1.2 or later)

Parameters

line
Line number of line to check.

Remarks

The **CHKSW** statement is useful to check if programs can run on the shareware release of NBASIC since some statements and functions (e.g. graphics) are only available in the standard release.

If *line* is omitted all lines are checked.

Examples

```
CHKSW
CHKSW 100
CHKSW 10-
CHKSW 10-100
```

```
CHKSW -100
CHKSW -10,20,50-100,900-
```

Mode

Immediate only

7.31 CHKSYN Statement (standard version only)

Checks for lines in the current program that contain syntax errors.

CHKSYN [*line*][**-**[*line*]]

- or -

CHKSYN [*line*][**-**[*line*]][,[*line*][**-**[*line*]]]... (Version 1.2 or later)

Parameters

line
Line number of line to check.

Remarks

The **CHKSYN** statement is useful to check programs for syntax errors that may not be found by running the program.

If *line* is omitted all lines are checked.

Examples

```
CHKSYN
CHKSYN 100
CHKSYN 10-
CHKSYN 10-100
CHKSYN -100
CHKSYN -10,20,50-100,900-
```

Mode

Immediate only

7.32 CHKUL Statement (standard version only)

Checks for lines in the current program that contain undefined line numbers.

CHKUL [*line*][**-**[*line*]]

- or -

CHKUL [*line*][**-**[*line*]][,[*line*][**-**[*line*]]]... (Version 1.2 or later)

Parameters

line
Line number of line to check.

Remarks

The **CHKUL** statement is useful to check programs for undefined line numbers that may not be found by running the program.

If *line* is omitted all lines are checked.

Examples

```
CHKUL
CHKUL 100
CHKUL 10-
CHKUL 10-100
CHKUL -100
CHKUL -10,20,50-100,900-
```

Mode

Immediate only

7.33 CHORD Statement (standard version only)

Draws a chord.

CHORD (*x***,***y***),***radius*[,[*color*][,[*start*][,[*end*][,[*aspect*][,**F**]]]]]

Parameters

x
X coordinate of center of chord.

y
Y coordinate of center of chord.

radius
Radius of chord.

color
Color of chord.

start
Starting angle of chord in radians.

end
Ending angle of chord in radians.

aspect
Ratio of length of y-axis to length of x-axis used to draw elliptical chords.

Remarks

The **CHORD** statement can be used to draw a chord of a circle or ellipse. A chord is an arc specified by the *start* and *end* angles with the endpoints connected.

Use **,F** to fill in the chord.

If *color* is omitted the default is the current graphics foreground color. If *start* is omitted the default is 0. If *end* is omitted the default is 0. if *aspect* is omitted the default is 1.

Graphics must be turned on to draw chords.

Examples

```
CHORD (50,50),25,,.785,3.142
CHORD (100,100),50,5,,2.356,,F
```

Mode

Program only

See Also

BOX, **CIRCLE**, **DRAW**, **GRAPH**, **LINE**, **PAINT**, **PCOLOR**, **PIE**

7.34 CIRCLE Statement (standard version only)

Draws a circle.

CIRCLE (*x***,***y***),***radius*[**,**[*color*][**,**[*start*][**,**[*end*][**,**[*aspect*][**,F**]]]]]

Parameters

x
X coordinate of center of circle.

y
Y coordinate of center of circle.

radius
Radius of circle.

color
Color of circle.

start
Starting angle of arc in radians.

end
Ending angle of arc in radians.

aspect
Ratio of length of y-axis to length of x-axis used to draw ellipses.

Remarks

The **CIRCLE** statement can be used to draw circles, ellipses, and arcs. To draw a circle *aspect* must be 1 or omitted. To draw ellipses *aspect* can be less than 1 for ellipses stretched on the x-axis or greater than 1 for ellipses stretched on the y-axis. To draw arcs specify the *start* and *end* angles.

Use **,F** to fill in the circle.

If *color* is omitted the default is the current graphics foreground color. If *start* is omitted the default is 0. If *end* is omitted the default is 0. if *aspect* is omitted the default is 1.

Graphics must be turned on to draw circles.

Examples

```
CIRCLE (100,100),50
CIRCLE (200,200),10,5,,,,F
CIRCLE (150,150),100,,,,2
CIRCLE (150,150),100,,,,.5
CIRCLE (50,50),25,,.785,3.142
```

Mode

Program only

See Also

BOX, **CHORD**, **DRAW**, **GRAPH**, **LINE**, **PAINT**, **PCOLOR**, **PIE**

7.35 CLEAR Statement

Clears one or more array variables.

CLEAR

- or -

CLEAR *variable*[**,***variable*]...

Parameters

variable
Array variable to clear.

Remarks

The **CLEAR** statement resets all elements of numeric array variables to 0 and string array variables to "".

CLEAR with no parameters clears all array variables.

Examples

```
CLEAR
CLEAR ENTRIES,USERS$
```

Mode

Immediate, Program

See Also

ERASE

7.36 CLOSE Statement

Closes one or more open files or the printer.

CLOSE

- or -

CLOSE #*file*[**,**#*file*]...

Parameters

file
File number of open file or printer (-2) to close.

Remarks

The **CLOSE** statement writes any remaining data in the file buffer to the file and closes the file.

CLOSE with no parameters closes all open files.

CLOSE #-2 ends the print job and closes the printer.

Examples

```
CLOSE
CLOSE #1
```

Mode

Immediate, Program

See Also

OPEN, **RESET**, **UNLOAD**

7.37 CLOSE PRINTER Statement (Version 1.1 or later, standard version only)

Closes the printer.

CLOSE PRINTER

Remarks

The **CLOSE PRINTER** statement ends the print job and closes the printer.

Mode

Program only

See Also

OPEN PRINTER

7.38 CLR Statement

Clears a row or clears variables.

CLR *row*[**,**[*column*][**,**[*columns*][**,***rows*]]]

- or -

CLR (Version 1.5 or later)

Parameters

row
Starting row.

column
Starting column.

columns
Number of columns to clear.

rows
Number of rows to clear.

Remarks

The **CLR** statement clears a number of rows and columns using the current background color and sets the cursor position to *row*,*column*.

If *column* is omitted the default is 0. if *columns* is omitted the default is the last column. If *rows* is omitted the default is 1.

CLR with no parameters resets numeric variables to 0, string variables to "" and deletes numeric and string array variables.

Examples

```
CLR 5
CLR 1,0,5,2
CLR 0,,10
CLR
```

Mode

Immediate, Program

See Also

CLS

7.39 CLS Statement

Clears the screen.

CLS

Remarks

The **CLS** statement clears the screen using the current background color and sets the cursor position to 0,0.

Mode

Immediate, Program

See Also

CLR

7.40 CMD Statement (Version 1.5 or later)

Copies screen output to a file.

CMD *file*[**,***expression*]

Parameters

file
File number of open file or printer (-2) to copy output to.

expression
Data to copy to file.

Remarks

The **CMD** statement causes screen output to be copied to a file or printer and prints the value of expression to the file or printer specified.

Use **PRINT** *#file* to stop screen output from being copied.

The **CMD** statement is similar to the **ATTACH** statement and is provided for compatibility.

Examples

```
CMD 1
CMD 1,"Test Data"
PRINT #1
```

Mode

Immediate, Program

See Also

ATTACH, **OPEN**, **PRINT**

7.41 COLOR Statement

Sets the screen display colors.

COLOR *foreground*[**,***background*]

Parameters

foreground
Foreground text color.

background
Background text color.

Remarks

The **COLOR** statement can be used to set the foreground and background text colors.

If *background* is omitted the background color is not changed.

Examples

```
COLOR 12
COLOR 0,10
```

Mode

Immediate, Program

See Also

INVERSE, **COLOR**

7.42 COLOR Statement, graphics (Version 1.4 or later, standard version only)

Sets the graphics foreground color.

COLOR=*foreground*

Parameters

foreground
Foreground graphics color.

Remarks

The **COLOR** statement can be used to set the foreground graphics colors.

Graphics must be turned on to set the graphics colors.

The **COLOR** statement is similar to the **PCOLOR** statement and is provided for compatibility.

Examples

```
COLOR=4
```

Mode

Program only

See Also

HCOLOR, **PCOLOR**

7.43 CONCAT Statement (Version 1.5 or later, standard version only)

Appends a file to another file.

CONCAT *filespec1* **TO** *filespec2*

Parameters

filespec1
Name of existing file.

filespec2
Name of new or existing file.

Remarks

The **CONCAT** statement can be used to append an existing file to another file.

The **CONCAT** statement is equivalent to the **APPEND** statement and is provided for compatibility.

Examples

```
CONCAT "TEST.DAT" TO "CONFIG.DAT"
CONCAT "PROGRAMS:TEST.DAT" TO "DATA:TEST.DAT"
```

Mode

Immediate, Program

See Also

APPEND

7.44 CONFIRM Statement

Turns confirmation of statements that may erase data on or off.

CONFIRM {**ON** | **OFF**}

Remarks

If confirmation is turned on, statements that may erase or overwrite data display a prompt allowing the operation to continue or be cancelled.

Only statements that are executed in immediate mode will display a prompt.

Mode

Immediate only

Examples

```
CONFIRM ON
CONFIRM OFF
```

See Also

SECURE

7.45 CONT Statement

Continues execution of the current program after a break.

CONT

Remarks

The **CONT** statement can be used to continue a program that has been stopped with the BREAK key, a **STOP** statement or a breakpoint (standard version only).

If any changes are made to the program after a break, the **CONT** statement cannot continue execution. The program will have to be restarted using the **RUN** statement.

Mode

Immediate only

See Also

RUN, **STOP**

7.46 CONTINUE Statement (Version 1.4 or later)

Continues execution of the current program after a break.

CON[**TINUE**]

Remarks

The **CONTINUE** statement can be used to continue a program that has been stopped with the BREAK key, a **STOP** statement or a breakpoint (standard version only).

If any changes are made to the program after a break, the **CONTINUE** statement cannot continue execution. The program will have to be restarted using the **RUN** statement.

The **CONTINUE** statement is equivalent to the **CONT** statement and is provided for compatibility.

Mode

Immediate only

Examples

```
CONTINUE
CON
```

See Also

CONT

7.47 COPY Statement

Copies a file to another file.

COPY *filespec1* **TO** *filespec2*

- or -

C. *filespec1* **TO** *filespec2* (Version 1.5 or later)

Parameters

filespec1
Name of existing file.

filespec2

Name of new file.

Remarks

The **COPY** statement can be used to copy an existing file to another file or volume.

The *filespec* and *filespec2* parameters may contain the wildcard characters * and ?. (Version 1.5 or later)

Examples

```
COPY "TEST.DAT" TO "CONFIG.DAT"
COPY "PROGRAMS:TEST.DAT" TO "DATA:TEST.DAT"
COPY "*.DOC" TO "TEMP:"
COPY "*.TXT" TO "*.DOC"
COPY "*.LOG" TO "HISTORY.TXT"
```

Mode

Immediate, Program

See Also

APPEND, **MOVE**, **RENAME**

7.48 COPY Statement, editing (standard version only)

Copies a line in the current program to another line.

COPY *line1* **TO** *line2*[**,E**]

- or -

C. *line1* **TO** *line2*[**,E**] (Version 1.5 or later)

Parameters

line1
Line number of line to copy.

line2
Line number of destination line.

Remarks

The **COPY** statement copies an existing line in the current program to another line. The destination line is overwritten if it already exists. Line numbers in the destination line that refer to the line being copied are automatically changed to the destination line number.

Use **,E** to edit the destination line.

Examples

```
COPY 10 TO 50
COPY 20 TO 100,E
```

Mode

Immediate only

See Also

APPEND, **MOVE**, **SWAP**

7.49 CREATE Statement

Creates a file.

CREATE *filespec*

- or -

CR. *filespec* (Version 1.5 or later)

Parameters

filespec
Name of file to create.

Remarks

The **CREATE** statement creates an empty file. If the file exists it is truncated to 0 length.

The default extension for the **CREATE** statement if not specified in the *filespec* is "dat".

Examples

```
CREATE "TEST.DAT"
```

Mode

Immediate, Program

7.50 CURSOR Statement

Turns the cursor on or off.

CURSOR {**ON** | **OFF**}

Examples

```
CURSOR ON
CURSOR OFF
```

Mode

Immediate, Program

See Also

LOCATE

7.51 DATA Statement

Specifies values to be read by subsequent **READ** statements.

DATA *value*[**,***value*]...

Parameters

value
A numeric or string constant.

Remarks

String constants that contain commas (,), colons (:), leading or trailing spaces should be enclosed in quotation marks (" ").

DATA statements can be placed anywhere within a program except after a **REM** statement.

Examples

```
DATA 1,2,3,A,B,C
DATA "Hello, world!",1000
DATA &H1F,&O377,&B1001
DATA -1.2345E10,9.876E-12
```

Mode

Program only

See Also

READ, **RESTORE**

7.52 DEBUG Statement (standard version only)

Turns debugging on or off.

DEBUG {**ON** | **OFF**}

Remarks

The **DEBUG** statement can be used to enable or disable debugging when a program is run. After setting breakpoints turn debugging on. When the program reaches a breakpoint, it will stop. To run the program without stopping at breakpoints turn debugging off rather than clearing breakpoints.

Examples

```
DEBUG ON
DEBUG OFF
```

Mode

Immediate, Program

See Also

ASSERT, **BREAK**, **STEP**

7.53 DEC Statement (standard version only)

Decrements one or more numeric variables by 1.

DEC *variable*[**,***variable*]...

Parameters

variable
Numeric variable to decrement.

Examples

```
DEC A
DEC ITEMS,USERS
```

Mode

Immediate, Program

See Also

INC

7.54 DEF FN Statement

Defines a function.

DEF FN *function*[**(***parameterlist***)**]**=***expression*

Parameters

function
Name of function.

parameterlist
One or more arguments.

expression
Return value of function.

Remarks

The **DEF FN** statement can be used to create a user-defined numeric function. The user-defined function can have up to ten parameters in the parameter list. The parameters can be numeric or string.

The *expression* must evaluate to a numeric value. If *expression* contains other user-defined functions, they must be defined prior to the **DEF FN** statement that calls them.

Examples

```
DEF FN DOUBLE(X)=X*2
DEF FN E=2.718281828459
DEF FN CENTER(A$)=(80-LEN(A$))/2
DEF FN AVG2(X,Y)=(X+Y)/2
```

Mode

Program only

See Also

FN

7.55 DEL Statement

Deletes lines from the current program.

DEL {*line*[**-***line*] | *-line*}

- or -

DEL {*line*[**-***line*] | *-line*}[,{*line*[**-***line*] | *-line*}]... (Version 1.2 or later)

Parameters

line
Line number of line to delete.

Examples

```
DEL 100
DEL 10-90
DEL -200
DEL -10,20,50-100,900-
```

Mode

Immediate only

7.56 DELETE Statement

Deletes a record from a file.

DELETE #*file*,*record*

Parameters

file
File number of open random access file.

record
Record number of record to delete.

Remarks

The **DELETE** statement deletes a record from a random access file, moves subsequent records down, and truncates the file.

Example

```
DELETE #1,5
```

Mode

Immediate, Program

See Also

INSERT, **OPEN**

7.57 DETACH Statement (Version 1.5 or later)

Detaches screen output to a file.

DETACH {*#file* | **LPRINT**}

Parameters

file
File number of open file or printer (-2) to detach.

Remarks

The **DETACH** statement stops screen output from being copied to a file or a printer.

Examples

```
DETACH #1
DETACH LPRINT
```

Mode

Immediate, Program

See Also

ATTACH, **OPEN**

7.58 DIM Statement

Declares an array.

DIM *array***(***subscripts***)**[**,***array***(***subscripts***)**]...

Parameters

array
Name of array.

subscripts
Dimensions of array.

Remarks

An array may have up to ten dimensions and 65536 elements per dimension (including element 0). However, the maximum size of the array including all dimensions cannot exceed 16777216 elements. The maximum size of the array is also constrained by available memory.

Elements of numeric arrays are initialized to 0 and elements of string arrays are initialized to "".

If **OPTION BASE 1** is specified, array elements begin at index 1 not 0.

Examples

```
DIM A(20,20)
DIM B$(100),C(5,20)
```

Mode

Immediate, Program

See Also

ERASE, **OPTION BASE**, **OPTION EXPLICIT**, **REDIM**

7.59 DIR Statement

Lists files on a volume.

DIR [*filespec*]

Parameters

filespec
Files to list.

Remarks

The **DIR** statement lists the files on a volume and information about each file.

The *filespec* parameter may contain the wildcard characters * and ?. If the *filespec* contains only a volume name the colon (:) must be included. If the *filespec* does not include a volume name, the default volume is used.

If *filespec* is omitted, files on the default volume are listed.

Examples

```
DIR
DIR "TEST.*"
DIR "*.DAT"
DIR "EXAMPLES:"
```

Mode

Immediate only

See Also

VOLUME, **VOLUMES**

7.60 DIRR Statement

Lists built-in programs.

DIRR

Mode

Immediate only

See Also

LOADR, **RUNR**

7.61 DISPLAY Statement (Version 1.5 or later)

Writes output to the screen.

DISPLAY [*expression*] [{**,** | **;** | **SPC(***spaces***)** | **TAB(***column***)**} [*expression*]]...

Parameters

expression
Data to print.

spaces
Number of spaces to output.

column
Column where output is to start.

Remarks

A **,** (comma) starts printing at the next print zone. Print zones are 10 characters wide. A **;** starts printing immediately after the last value printed. **SPC** prints a specified number of spaces. **TAB** starts printing at the specified column.

If any delimiters (**,** | **;** | **SPC** | **TAB**) are not specified as the last item in the print statement a carriage return is printed erasing the rest of the line and moving the print position to the beginning of the next line.

The **DISPLAY** statement is similar to the **PRINT** statement and is provided for compatibility.

Examples

```
DISPLAY
DISPLAY "HELLO"
DISPLAY "ABC","123"
DISPLAY 100;" APPLES"
DISPLAY "TOTAL:" SPC(5) T
DISPLAY #1, "ITEMS:";I
```

Mode

Immediate, Program

See Also

PRINT

7.62 DLOAD Statement (Version 1.4 or later)

Loads a string from a file.

DLOAD *filespec***,***variable*

Parameters

filespec
Name of file to load string from.

variable
Variable to load file into.

Remarks

The **DLOAD** statement can be used to load the contents of a file into a string variable.

The maximum file size that can be loaded is limited to the maximum string length which is 65535 characters.

The default extension for the **DLOAD** statement if not specified in the *filespec* is "dat".

Examples

```
DLOAD "NAMES.TXT",N$
```

Mode

Immediate, Program

See Also

DSAVE

7.63 DRAW Statement (standard version only)

Draws objects.

DRAW *command*

Parameters

command
String containing draw commands.

Remarks

Draw commands:

D[*n*]	Moves graphics cursor down *n* pixels.
E[*n*]	Moves graphics cursor up and right *n* pixels.
F[*n*]	Moves graphics cursor down and right *n* pixels.
G[*n*]	Moves graphics cursor down and left *n* pixels.
H[*n*]	Moves graphics cursor up and left *n* pixels.
L[*n*]	Moves graphics cursor left *n* pixels.
M[{+ | -}]*x*,*y*	Moves graphics cursor to point *x*,*y*. If *x* is preceded by + or -, moves relative to the current graphics point.
R[*n*]	Moves graphics cursor right *n* pixels.
U[*n*]	Moves graphics cursor up *n* pixels.
[B]	Optional prefix that moves graphics cursor without drawing.

[N]	Optional prefix that draws and returns graphics cursor to original position.
A*n*	Sets the drawing angle (0-3) (Version 1.4 or later).
C*n*	Sets the drawing color (0-15).
S*n*	Sets the drawing scale (1-255) (Version 1.4 or later).
P*f*,*b*	Paints a graphics area using color *f*. Painting stops at border color *b*.
Xv$	Draws a substring (Version 1.4 or later).
{; | *space*}	Separates drawing commands.

The drawing angle rotates the object in increments of 90 degrees.

To draw a substring use the VARPTR$ function to get the address of the string variable containing the substring draw.

Graphics must be turned on to draw objects.

Examples

```
DRAW "M200,200;D50;R50;U50;L50;BM225,225;P10,0"
A$="D50;R50;U50;L50;"
DRAW "M200,200;X"+VARPTR$(A$)+"BM225,225;P10,0"
```

Mode

Program only

See Also

BOX, **CHORD**, **CIRCLE**, **GRAPH**, **LINE**, **PAINT**, **PCOLOR**, **PIE**, **VARPTR$**

7.64 DSAVE Statement (Version 1.4 or later)

Saves a string to a file.

DSAVE *filespec*,*string*

Parameters

filespec
Name of file to save string to.

string
String to save.

Remarks

The **DSAVE** statement can be used to save a string to a file.

The longest string that can be saved is limited by the maximum string length which is 65535 characters.

The default extension for the **DSAVE** statement if not specified in the *filespec* is "dat".

Examples

```
DSAVE "NAMES.TXT",N$
```

Mode

Immediate, Program

See Also

DLOAD

7.65 DUMP Statement

Displays the contents of a file.

DUMP *filespec*[**,***format*]

Parameters

filespec
Name of file to dump.

format
Format of display.

Remarks

The *format* parameter may be A (ascii), O (octal) or D (decimal).

If *format* is omitted, the default is hexadecimal.

Examples

```
DUMP "TEST.DAT"
DUMP "TEST.DAT",A
```

Mode

Immediate only

See Also

TYPE

7.66 EDIT Statement

Edits a line in the current program.

EDIT *line*

- or -

EDIT {*line*[**-***line*] | *-line*}[,{*line*[**-***line*] | *-line*}]... (Version 1.4 or later)

Parameters

line
Line number of line to edit.

Example

```
EDIT 100
```

```
EDIT 10-90
EDIT -200
EDIT -10,20,50-100,900-
```

Mode

Immediate only

7.67 END Statement

Ends a program.

END

Remarks

The **END** statement ends the currently executing program or immediate mode command line, closes all open files and closes the printer.

Mode

Immediate, Program

See Also

STOP

7.68 ERASE Statement

Deletes one or more array variables.

ERASE

- or -

ERASE *variable*[**,***variable*]...

Parameters

variable
Array variable to erase.

Remarks

The **ERASE** statement deletes an array variable and all of its elements. The array variable can subsequently be used in a **DIM** statement.

ERASE with no parameters deletes all array variables.

Examples

```
ERASE
ERASE ENTRIES,USERS$
```

Mode

Immediate, Program

See Also

CLEAR, **DIM**

7.69 ERROR Statement

Simulates an occurrence of an error.

ERROR *error*

Parameters

error
Error number.

Remarks

The **ERROR** statement can be used to test error handling within a program or to issue user defined errors. User defined errors are in the range of 128 to 255.

Examples

```
ERROR 24
ERROR 128
```

Mode

Immediate, Program

See Also

ERL, **ERR**, **ERR$**, **ON ERROR**

7.70 EXEC Statement

Executes an internal routine.

EXEC *address*

Parameters

address
Address of internal routine.

Examples

```
EXEC &H9305
```

Mode

Immediate, Program

See Also

PEEK, **POKE**

7.71 EXIT Statement

Exits NBASIC.

EXIT

Remarks

The **EXIT** statement exits NBASIC and prompts to save the current program if modified.

Mode

Immediate only

7.72 FIELD Statement

Allocates space for variables in a random access file buffer.

FIELD #*file*,*width* **AS** *variable*[,*width* **AS** *variable*]...

Parameters

file
File number of open random access file.

width
Number of characters in field.

variable
String variable that identifies the field and contains field data.

Remarks

If multiple random access files are open at the same time, specify different field variables for each file. Do not use field variables in other assignment statements other than **LSET** and **RSET**, the variable will no longer identify the field.

If numeric values will be stored in a field in binary format, the field must be allocated with a width of 8. Use the **MKN$** function to convert a number to a binary string before storing the data in a field.

Example

```
FIELD #1,8 AS ID$,32 AS USER$,12 AS PHONE$
```

Mode

Immediate, Program

See Also

GET, **LSET**, **OPEN**, **PUT**, **RSET**, **CVN**, **MKN$**

7.73 FILES Statement

Lists files on a volume.

FILES [*filespec*]

Parameters

filespec
Files to list.

Remarks

The **FILES** statement lists the files on a volume and information about each file.

The *filespec* parameter may contain the wildcard characters * and ?. If the *filespec* contains only a volume name the colon (:) must be included. If the *filespec* does not include a volume name, the default volume is used.

If *filespec* is omitted, files on the default volume are listed.

The **FILES** statement is equivalent to the **DIR** statement and is provided for compatibility.

Examples

```
FILES
FILES "TEST.*"
FILES "*.DAT"
FILES "EXAMPLES:"
```

Mode

Immediate only

See Also

DIR

7.74 FILL Statement (standard version only)

Fills a row with a character.

FILL *char*,*row*[,[*column*][,[*columns*][,*rows*]]]

Parameters

char
Character to fill row with.

row
Starting row.

column
Starting column.

columns
Number of columns to fill.

rows
Number of rows to fill.

Remarks

The **FILL** statement fills a number of rows and columns with the specified character using the current foreground and background colors.

If *column* is omitted the default is 0. if *columns* is omitted the default is the last column. If *rows* is omitted the default is 1.

Examples

```
FILL "*",5
FILL "@",1,0,5,2
FILL "+",0,,10
FILL ">",10,5,1,8
```

Mode

Immediate, Program

7.75 FIND Statement (Version 1.1 or later, standard version only)

Finds a string in the current program.

FIND {*line*[**-***line*] | *-line*}

- or -

FIND {*line*[**-***line*] | *-line*}[,{*line*[**-***line*] | *-line*}]... (Version 1.2 or later)

Parameters

string
String to find.

line
Line number of line to search in.

Remarks

The **FIND** statement searches for *string* in the current program. If a match is found, the line containing the string is displayed with each occurrence highlighted.

The search is not case sensitive.

If *line* is omitted the entire current program is searched.

Example

```
FIND "INPUT"
FIND "PRINT",100-200
FIND "A=",-100,1000-2000
```

Mode

Immediate only

7.76 FONT Statement (standard version only)

Changes the font size.

FONT *size*[**,R**]

Parameters

size
Size of font (in points).

Remarks

The **FONT** statement changes the size of the screen font to the size specified and clears the screen.

Use **,R** to resize the NBASIC window.

Examples

```
FONT 10
FONT 12,R
```

Mode

Immediate only

7.77 FOR Statement

Repeats a block of statements a specified number of times.

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

Parameters

counter
Numeric variable used as loop counter.

start
Initial value of counter.

end
Final value of counter.

increment
Amount counter is changed each time through the loop.

Remarks

Each **FOR** statement must have a matching **NEXT** statement.

If **STEP** *increment* is omitted the default is 1. To decrement a counter specify a negative *increment*.

NOTE: A **FOR** loop is always executed at least once regardless of the *end* and *increment* values.

Examples

```
FOR I=1 TO 10
FOR J=2 TO 100 STEP 2
FOR K=10 TO 1 STEP -1
```

Mode

Immediate, Program

See Also

NEXT

7.78 FORMAT Statement (Version 1.4 or later, standard version only)

Formats the current program.

FORMAT {[**NO**]**CAP** | [**NO**]**LET**}

Remarks

The **FORMAT** statement changes a program to upper (CAP) or lower (NOCAP) case or the **LET** statement to show (LET) or hide (NOLET) the LET keyword.

Examples

```
FORMAT CAP
FORMAT NOCAP
FORMAT LET
FORMAT NOLET
```

Mode

Immediate only

See Also

LET

7.79 FRAME Statement

Draws a frame.

FRAME *row*,*column*,*columns*,*rows*[,[*style*][,*color*]]

Parameters

row
Starting row.

column
Starting column.

columns
Number of columns.

rows
Number of rows.

style
Style of frame.

color
Color of frame.

Remarks

The **FRAME** statement draws a frame *columns* wide and *rows* high beginning at *row*,*column*. The *style* of the frame can be either block (0) or line (1). The interior of the frame is cleared using the current background text color.

If *style* is omitted the default is 0. If *color* is omitted the default is the current foreground text color.

Examples

```
FRAME 1,2,76,10
FRAME 1,2,76,10,1
FRAME 1,2,76,10,1,12
FRAME 1,2,76,10,,12
```

Mode

Immediate, Program

7.80 FRE Statement

Initializes the run-time environment.

FRE

Remarks

The **FRE** statement is equivalent to the **INIT** statement and is provided for compatibility.

Mode

Immediate, Program

See Also

INIT

7.81 GET Statement

Reads a record from a random access file into a file buffer.

GET #*file*,*record*

Parameters

file
File number of open random access file.

record
Record number of record to get.

Example

```
GET #1,1
```

Mode

Immediate, Program

See Also

FIELD, **LSET**, **OPEN**, **PUT**, **RSET**

7.82 GET Statement, graphics (standard version only)

Captures a graphics screen image.

GET (*x1***,***y1***)-(***x2***,***y2***),***variable*

Parameters

x1
X coordinate of corner of graphics image.

y1
Y coordinate of corner of graphics image.

x2
X coordinate of opposite corner of graphics image.

y2
Y coordinate of opposite corner of graphics image.

variable
Array variable where image is stored.

Remarks

Each pixel in the image requires one element of array storage. The size of the array can be found by using the equation (*x2*-*x1*+1) * (*y2*-*y1*+1), where (*x1*,*y1*) is the upper left corner of the image and (*x2*,*y2*) is the lower right corner.

Graphics must be turned on to capture images.

Examples

```
GET (10,10)-(20,20),IMAGE1
```

Mode

Program only

See Also

GRAPH, **PUT**

7.83 GOSUB Statement

Transfers control of the current program to a subroutine.

GOSUB *line*

Parameters

line
Line number or label of subroutine.

Remarks

The **GOSUB** statement transfers control to the specified subroutine. Use the **RETURN** statement to return from the subroutine.

Examples

```
GOSUB 100
GOSUB *START
```

Mode

Immediate, Program

See Also

GOTO, **ON GOSUB**, **POP**, **PUSH**, **RETURN**

7.84 GOTO Statement

Transfers control of the current program to a specified line.

GOTO *line*

Parameters

line
Line number or label of line.

Examples

```
GOTO 100
GOTO *START
```

Mode

Immediate, Program

See Also

GOSUB, **ON GOTO**

7.85 GOTO TIMER Statement

Transfers control of the current program to the timer event handling routine.

GOTO TIMER

Remarks

The **GOTO TIMER** statement transfers control to the timer handler just as if an actual timer event had occurred. The timer handler must be set using the **ON TIMER** statement. Control will be returned to the current program as specified by the **RESUME** statement in the timer handler.

Example

```
GOTO TIMER
```

Mode

Program

See Also

ON TIMER, **RESUME**

7.86 GR Statement (Version 1.4 or later, standard version only)

Turns graphics on.

GR

Remarks

In order to draw graphics on the screen graphics must be enabled.

The **GR** statement is equivalent to the **GRAPH ON** statement and is provided for compatibility.

Mode

Program only

See Also

GRAPH, **HGR**, **HGR2**, **TEXT**

7.87 GRAPH Statement (standard version only)

Turns graphics on or off.

GRAPH {[**ON**] | **OFF**}

Remarks

In order to draw graphics on the screen graphics must be enabled.

Examples

```
GRAPH
GRAPH ON
GRAPH OFF
```

Mode

Program only

See Also

BOX, **CHORD**, **CIRCLE**, **DRAW**, **GET**, **LINE**, **PAINT**, **PCLR**, **PCLS**, **PCOLOR**, **PFONT**, **PIE**, **POINT**, **PPRINT**, **PRESET**, **PSET**, **PUT**

7.88 HCIRCLE Statement (Version 1.5 or later, standard version only)

Draws a circle.

HCIRCLE (*x***,***y***),***radius*[**,**[*color*][**,**[*start*][**,**[*end*][**,**[*aspect*][**,F**]]]]]

Parameters

x
X coordinate of center of circle.

y
Y coordinate of center of circle.

radius
Radius of circle.

color
Color of circle.

start
Starting angle of arc in radians.

end
Ending angle of arc in radians.

aspect
Ratio of length of y-axis to length of x-axis used to draw ellipses.

Remarks

The **HCIRCLE** statement can be used to draw circles, ellipses, and arcs. To draw a circle *aspect* must be 1 or omitted. To draw ellipses *aspect* can be less than 1 for ellipses stretched on the x-axis or greater than 1 for ellipses stretched on the y-axis. To draw arcs specify the *start* and *end* angles.

Use **,F** to fill in the circle.

If *color* is omitted the default is the current graphics foreground color. If *start* is omitted the default is 0. If *end* is omitted the default is 0. if *aspect* is omitted the default is 1.

Graphics must be turned on to draw circles.

The **HCIRCLE** statement is equivalent to the **CIRCLE** statement and is provided for compatibility.

Examples

```
HCIRCLE (100,100),50
HCIRCLE (200,200),10,5,,,,F
HCIRCLE (150,150),100,,,,2
HCIRCLE (150,150),100,,,,.5
HCIRCLE (50,50),25,,.785,3.142
```

Mode

Program only

See Also

CIRCLE, **GRAPH**, **HCLS**, **HCOLOR**, **HDRAW**, **HLINE**, **HPAINT**

7.89 HCLS Statement (Version 1.5 or later, standard version only)

Clears the graphics screen.

HCLS [*color*]

Parameters

color
Color to clear screen with.

Remarks

The **HCLS** statement clears the graphics screen with *color* and sets the graphics background color to *color*.

If *color* is omitted the default is the current graphics background color.

Graphics must be turned on to clear the graphics screen.

The **HCLS** statement is equivalent to the **PCLS** statement and is provided for compatibility.

Examples

```
HCLS
HCLS 7
```

Mode

Program only

See Also

HCOLOR, **PCLS**

7.90 HCOLOR Statement (Version 1.4 or later, standard version only)

Sets the graphics foreground color.

HCOLOR=*foreground*

- or -

HCOLOR *foreground*[**,***background*] (Version 1.5 or later, standard version only)

Parameters

foreground
Foreground graphics color.

background
Background graphics color.

Remarks

The **HCOLOR** statement can be used to set the foreground graphics colors.

Graphics must be turned on to set the graphics colors.

The **HCOLOR** statement is similar to the **PCOLOR** statement and is provided for compatibility.

Examples

```
HCOLOR=4
HCOLOR 4
HCOLOR 7,10
```

Mode

Program only

See Also

COLOR, **PCOLOR**

7.91 HDRAW Statement (Version 1.5 or later, standard version only)

Draws objects.

HDRAW *command*

Parameters

command
String containing draw commands.

Remarks

Draw commands:

D[*n*]	Moves graphics cursor down *n* pixels.
E[*n*]	Moves graphics cursor up and right *n* pixels.
F[*n*]	Moves graphics cursor down and right *n* pixels.
G[*n*]	Moves graphics cursor down and left *n* pixels.
H[*n*]	Moves graphics cursor up and left *n* pixels.
L[*n*]	Moves graphics cursor left *n* pixels.
M[{+ | -}]*x*,*y*	Moves graphics cursor to point *x*,*y*. If *x* is preceded by + or -, moves relative to the current graphics point.
R[*n*]	Moves graphics cursor right *n* pixels.
U[*n*]	Moves graphics cursor up *n* pixels.
[B]	Optional prefix that moves graphics cursor without drawing.
[N]	Optional prefix that draws and returns graphics cursor to original position.
A*n*	Sets the drawing angle (0-3).
C*n*	Sets the drawing color (0-15).
S*n*	Sets the drawing scale (1-255).
P*f*,*b*	Paints a graphics area using color *f*. Painting stops at border color *b*.
Xv$	Draws a substring.
{; | *space*}	Separates drawing commands.

The drawing angle rotates the object in increments of 90 degrees.

To draw a substring use the VARPTR$ function to get the address of the string variable containing the substring draw.

Graphics must be turned on to draw objects.

The **HDRAW** statement is equivalent to the **DRAW** statement and is provided for compatibility.

Examples

```
HDRAW "M200,200;D50;R50;U50;L50;BM225,225;P10,0"
A$="D50;R50;U50;L50;"
HDRAW "M200,200;X"+VARPTR$(A$)+"BM225,225;P10,0"
```

Mode

Program only

See Also

DRAW, **VARPTR$**

7.92 HELP Statement

Displays help.

HELP [*topic*]

- or -

H. [*topic*] (Version 1.5 or later)

Parameters

topic
Topic to display help for.

Remarks

If *topic* exists in the help file, it is displayed. If more than one match is found then the help index is displayed.

If *topic* is omitted the default help topic is displayed.

Examples

```
HELP
HELP "RUN"
```

Mode

Immediate only

7.93 HGET Statement (Version 1.5 or later, standard version only)

Captures a graphics screen image.

HGET (*x1***,***y1***)-(***x2***,***y2***),***variable***

Parameters

x1
X coordinate of corner of graphics image.

y1
Y coordinate of corner of graphics image.

x2
X coordinate of opposite corner of graphics image.

y2
Y coordinate of opposite corner of graphics image.

variable
Array variable where image is stored.

Remarks

Each pixel in the image requires one element of array storage. The size of the array can be found by using the equation (*x2*-*x1*+1) * (*y2*-*y1*+1), where (*x1*,*y1*) is the upper left corner of the image and (*x2*,*y2*) is the lower right corner.

Graphics must be turned on to capture images.

The **HGET** statement is equivalent to the **GET** statement and is provided for compatibility.

Examples

```
HGET (10,10)-(20,20),IMAGE1
```

Mode

Program only

See Also

GRAPH, **GET**, **HPUT**

7.94 HGR Statement (Version 1.4 or later, standard version only)

Turns graphics on.

HGR

Remarks

In order to draw graphics on the screen graphics must be enabled.

The **HGR** statement is equivalent to the **GRAPH ON** statement and is provided for compatibility.

Mode

Program only

See Also

GR, **GRAPH**, **HGR2**, **TEXT**

7.95 HGR2 Statement (Version 1.4 or later, standard version only)

Turns graphics on.

HGR2

Remarks

In order to draw graphics on the screen graphics must be enabled.

The **HGR2** statement is equivalent to the **GRAPH ON** statement and is provided for compatibility.

Mode

Program only

See Also

GR, **GRAPH**, **HGR**, **TEXT**

7.96 HLIN Statement (Version 1.4 or later, standard version only)

Draws a horizontal line.

HLIN *x1*,*x2* **AT** *y*

Parameters

x1
X coordinate of start of line.

x1
X coordinate of end of line.

y
Y coordinate of line.

Remarks

The **HLIN** statement draws a line from the point (x1,y) to the point (x2,y) using the current graphics foreground color.

The **HLIN** statement is similar to the **LINE** statement and is provided for compatibility.

Graphics must be turned on to draw lines.

Examples

```
HLIN 10,100 AT 100
```

Mode

Program only

See Also

COLOR, **GR**, **LINE**, **VLIN**

7.97 HLINE Statement (Version 1.5 or later, standard version only)

Draws a line.

HLINE (*x1***,***y1***)-(***x2***,***y2***)[,**[*color*][,**B**[**F**]]]

- or -

HLINE -(*x2***,***y2***)[,**[*color*][,**B**[**F**]]]

Parameters

x1
X coordinate of starting point of line.

y1
Y coordinate of starting point of line.

x2
X coordinate of end point of line.

y2
Y coordinate of end point of line.

color
Color of line.

Remarks

The **LINE** statement can be used to draw lines or boxes. Form 1 draws a line from the point (*x1*,*y1*) to the point (*x2*,*y2*). Form 2 draws a line from the current graphics point to the point (*x2*,*y2*).

Use **,B** to draw a box. Use **F** to fill in the box.

If *color* is omitted the default is the current graphics foreground color.

Graphics must be turned on to draw lines.

The **HLINE** statement is equivalent to the **LINE** statement and is provided for compatibility.

Examples

```
HLINE (10,10)-(100,100)
HLINE (20,20)-(50,50),10
HLINE (200,200)-(250,250),,B
HLINE (220,220)-(230,230),12,BF
HLINE -(300,300)
HLINE -(100,100),5
HLINE -(80,80),2,BF
```

Mode

Program only

See Also

GRAPH, **LINE**

7.98 HOME Statement

Clears the screen.

HOME

Remarks

The **HOME** statement clears the screen using the current background color and sets the cursor position to 0,0.

The **HOME** statement is equivalent to the **CLS** statement and is provided for compatibility.

Mode

Immediate, Program

See Also

CLS

7.99 HPAINT Statement (Version 1.5 or later, standard version only)

Fills a graphics area with a color.

HPAINT (*x***,***y***)**[**,**[*color*][**,***border*]]

Parameters

x
X coordinate where painting begins.

y
Y coordinate where painting begins.

color
Color to paint.

border
Color of border where painting stops.

Remarks

If *color* is omitted the default is the current graphics foreground color. If *border* is omitted the default is the current graphics background color.

Graphics must be turned on to paint a graphics area.

The **HPAINT** statement is equivalent to the **PAINT** statement and is provided for compatibility.

Examples

```
HPAINT (100,100)
HPAINT (50,50),7
HPAINT (200,50),5,7
```

Mode

Program only

See Also

GRAPH, **PAINT**

7.100 HPLOT Statement (Version 1.4 or later, standard version only)

Draws a point or one or more lines.

HPLOT *x*,*y*

- or -

HPLOT *x1*,*y1* [**TO** *x2*,*y2*]...

- or -

HPLOT TO *x1*,*y1* [**TO** *x2*,*y2*]...

Parameters

x
X coordinate of point.

y
Y coordinate of point.

x1
X coordinate of starting point of line.

y1
Y coordinate of starting point of line.

x2
X coordinate of end point or line.

y2
Y coordinate of end point of line.

Remarks

The **HPLOT** statement draws a point or one or more lines at the specified coordinates using the current graphics foreground color.

If no starting point is specified the line is drawn from the current graphics point.

The **HPLOT** statement is similar to the **PSET** and **LINE** statements and is provided for compatibility.

Graphics must be turned on to draw points.

Examples

```
HPLOT 50,50
```

```
HPLOT 50,50 TO 100,100
HPLOT 50,50 TO 100,100 TO 10,10
HPLOT TO 50,50
HPLOT TO 50,50 TO 100,100 TO 10,10
```

Mode

Program only

See Also

COLOR, **GR**, **LINE**, **PLOT**, **PSET**

7.101 HPUT Statement (Version 1.5 or later, standard version only)

Displays a graphics screen image.

HPUT (*x1***,***y1***)-(***x2***,***y2***),***variable*[**,**{**AND** | **OR** | **PSET** | **PRESET** | **XOR**}]

Parameters

x1
X coordinate of corner of graphics image.

y1
Y coordinate of corner of graphics image.

x2
X coordinate of opposite corner of graphics image.

y2
Y coordinate of opposite corner of graphics image.

variable
Array variable where image is stored.

Remarks

Each pixel in the image requires one element of array storage. The size of the array can be found by using the equation (*x2*-*x1*+1) * (*y2*-*y1*+1), where (*x1*,*y1*) is the upper left corner of the image and (*x2*,*y2*) is the lower right corner.

AND mergers the stored image with the graphics screen.
OR superimposes the stored image on the graphics screen.
PSET draws stored image erasing the graphics screen.
PRESET draws stored image in reverse colors erasing the graphics screen.
XOR draws a stored image or erases a previously drawn image while preserving the background.

Graphics must be turned on to display images.

The **HPUT** statement is equivalent to the **PUT** statement and is provided for compatibility.

Examples

```
HPUT (110,110)-(120,120),IMAGE1
HPUT (210,210)-(220,220),IMAGE1,PSET
```

Mode

Program only

See Also

HGET, **GRAPH**, **PUT**

7.102 HRESET Statement (Version 1.5 or later, standard version only)

Draws a point.

HRESET (*x***,***y***)**[**,***color*]

Parameters

x
X coordinate of point.

y
Y coordinate of point.

color
Color of point.

Remarks

If *color* is omitted the default is the current graphics background color.

Graphics must be turned on to draw points.

The **HRESET** statement is equivalent to the **PRESET** statement and is provided for compatibility.

Examples

```
HRESET (50,50)
HRESET (100,100),5
```

Mode

Program only

See Also

GRAPH, **HSET**, **PRESET**

7.103 HSCROLL Statement (standard version only)

Scrolls columns left or right.

HSCROLL *row*,*column*,*columns*,*rows*[,*horizontal*]

Parameters

row
Starting row.

column
Starting column.

columns
Number of columns to scroll.

rows
Number of rows to scroll.

horizontal
Number of columns by which to scroll.

Remarks

The **HSCROLL** statement scrolls text within a rectangular region *length* columns by *height* rows starting at *row*,*column*, *horizontal* columns. If *horizontal* is positive the region is scrolled to the right, if negative to the left. The text is scrolled within the rectangle with columns to the right or left overwritten and new columns filled with spaces.

If *horizontal* is omitted the default is 1.

Examples

```
HSCROLL 0,0,80,25
HSCROLL 0,0,80,25,-1
```

Mode

Immediate, Program

See Also

VSCROLL

7.104 HSET Statement (Version 1.5 or later, standard version only)

Draws a point.

HSET (*x*,*y***)[,***color*]

Parameters

x
X coordinate of point.

y
Y coordinate of point.

color
Color of point.

Remarks

If *color* is omitted the default is the current graphics foreground color.

Graphics must be turned on to draw points.

The **HSET** statement is equivalent to the **PSET** statement and is provided for compatibility.

Examples

```
HSET (50,50)
HSET (100,100),5
```

Mode

Program only

See Also

GRAPH, **HRESET**, **PSET**

7.105 HTAB Statement (Version 1.4 or later)

Moves the cursor to a specified position on the screen.

HTAB *column*

Parameters

column
Column where cursor is to be moved.

Remarks

The **HTAB** statement only changes the column position of the cursor.

The screen has 80 columns numbered 0 through 79.

The **HTAB** statement is similar to the **LOCATE** statement and is provided for compatibility.

Examples

```
HTAB 10
```

Mode

Immediate, Program

See Also

LOCATE, **VTAB**

7.106 IF Statement

Executes one or more statements depending on specified conditions.

IF *condition* **THEN** {*line* | *statement*[*:statement*]...} [**ELSE** {*line* | *statement*[*:statement*]...}] [**END IF**]

- or -

IF *condition* **GOTO** *line* [**ELSE** {*line* | *statement*[*:statement*]...}] [**END IF**]

- or -

IF *condition* **GOSUB** *line* [**ELSE** {*line* | *statement*[*:statement*]...}] [**END IF**]

Parameters

condition
Expression that evaluates to true or false.

line
Line number or label of line.

statement
NBASIC statement.

Remarks

If *condition* is true, execution continues with the **THEN**, **GOTO**, or **GOSUB** clauses otherwise execution continues with the **ELSE** clause if specified.

The **END IF** clause ends the **IF** statement otherwise execution continues with statements following the **THEN** or **ELSE** clauses. It is useful when multiple statements are on the same line.

Examples

```
IF A<>0 THEN 100
IF N$="" THEN 50 ELSE 80
IF I/2>10 THEN I=I+1 ELSE I=0: GOTO 50 END IF
IF C>=100 GOTO 200
IF B(I)=0 GOSUB 100
```

Mode

Immediate, Program

See Also

ON GOTO, **ON GOSUB**

7.107 INC Statement (standard version only)

Increments one or more numeric variables by 1.

INC *variable*[**,***variable*]...

Parameters

variable
Numeric variable to increment.

Examples

```
INC A
INC ITEMS,USERS
```

Mode

Immediate, Program

See Also

DEC

7.108 INIT Statement

Initializes the run-time environment.

INIT

Remarks

The **INIT** statement deletes keystrokes in the type ahead buffer, deletes all variables, sets static variables to 0, and resets the **FILE$** and **VOLUME$** functions.

Mode

Immediate, Program

See Also

NEW

7.109 INPUT Statement

Reads input from the keyboard or a file.

INPUT [**;**] [*prompt*{**;** | **,**}] *variable*[,*variable*]...

- or -

INPUT #*file*,*variable*[,*variable*]...

Parameters

prompt
String displayed before data is entered. A semicolon after *prompt* appends a question mark (?) to the prompt string.

variable
Variable in which data read from keyboard or file is stored.

file
File number of open file.

Remarks

For keyboard input, a semicolon (;) immediately after **INPUT** keeps the cursor on the same line after the ENTER key is pressed.

Examples

```
INPUT C,N
INPUT "NAME"; N$
INPUT "NAME: ", N$
INPUT ; "TOTAL"; T
INPUT #1,N,A$
```

Mode

Program only

See Also

INPUT$, **LINE INPUT**, **OPEN**

7.110 INSERT Statement

Inserts a record in a file.

INSERT #*file*,*record*

Parameters

file
File number of open random access file.

record
Record number of record to insert.

Remarks

The **INSERT** statement extends a random access file, moves records up, and inserts a new record.

Example

```
INSERT #1,5
```

Mode

Immediate, Program

See Also

DELETE, **OPEN**

7.111 INVERSE Statement

Turns inverse on or off.

INVERSE {**ON** | **OFF**}

- or -

INVERSE (Version 1.4 or later)

Remarks

The **INVERSE** statement reverses the foreground and background colors.

Examples

```
INVERSE ON
INVERSE OFF
```

Mode

Immediate, Program

See Also

COLOR

7.112 INVERT Statement

Inverts a row.

INVERT *row*[**,**[*column*][**,**[*columns*][**,***rows*]]]

Parameters

row
Starting row.

column
Starting column.

columns
Number of columns to invert.

rows
Number of rows to invert.

Remarks

The **INVERT** statement inverts a number of rows and columns using the current foreground and background colors.

If *column* is omitted the default is 0. if *columns* is omitted the default is the last column. If *rows* is omitted the default is 1.

Examples

```
INVERT 5
INVERT 1,0,5,2
INVERT 0,,10
```

Mode

Immediate, Program

7.113 KILL Statement

Deletes a file.

KILL *filespec*

- or -

K. *filespec* (Version 1.5 or later)

Parameters

filespec
Name of file to delete.

Remarks

The **KILL** statement permanently deletes a file from a volume.

The default extension for the **KILL** statement if not specified in the *filespec* is "nba".

The *filespec* parameters may contain the wildcard characters * and ?. (Version 1.5 or later)

Examples

```
KILL "TEST.DAT"
KILL "DATA:NAMES.IDX"
KILL "*.TXT"
```

Mode

Immediate, Program

7.114 KILL # Statement (Version 1.1 or later)

Closes and deletes an open file.

KILL #*file*

Parameters

file
File number of open file or printer (-2) to kill.

Remarks

The **KILL #** statement writes any remaining data in the file buffer to the file, closes the file and deletes it.

KILL #-2 stops the print job and closes the printer.

Examples

```
KILL #1
```

Mode

Immediate, Program

See Also

OPEN, **RESET**, **UNLOAD**

7.115 LEFT$ Statement (standard version only)

Assigns part of a string variable to another string.

LEFT$(*variable*[**,***length*]**)=***expression*

Parameters

variable
String variable to assign string to.

length
Length of substring.

expression
String to assign.

Remarks

The **LEFT$** statement replaces the left *length* characters in *variable* with characters from *expression*.

If *length* is omitted the default is 1.

Examples

```
LEFT$(A$,4)="TEST"
LEFT$(B$)="0"
```

Mode

Immediate, Program

See Also

MID$**, **RIGHT$

7.116 LET Statement

Assigns the value of an expression to a variable.

[**LET**] *variable***=***expression*

- or -

[**LET**] *variable*[,*variable*]...**=***expression*[,*variable*[,*variable*]...**=***expression*]... (Version 1.2 or later)

Parameters

variable
Variable to assign value to.

expression
Value to assign to variable.

Examples

```
A=A+1
LET B=SQR(10)
LET N$="TEST"
A,B=1
LET A=1,B=2
```

Mode

Immediate, Program

7.117 LINE Statement (standard version only)

Draws a line.

LINE (*x1*,*y1*)**-(***x2*,*y2*)**[,**[*color*][,**B**[**F**]]]

- or -

LINE -(*x2*,*y2*)**[,**[*color*][,**B**[**F**]]]

Parameters

x1
X coordinate of starting point of line.

y1
Y coordinate of starting point of line.

x2
X coordinate of end point of line.

y2
Y coordinate of end point of line.

color
Color of line.

Remarks

The **LINE** statement can be used to draw lines or boxes. Form 1 draws a line from the point (*x1*,*y1*) to the point (*x2*,*y2*). Form 2 draws a line from the current graphics point to the point (*x2*,*y2*).

Use **,B** to draw a box. Use **F** to fill in the box.

If *color* is omitted the default is the current graphics foreground color.

Graphics must be turned on to draw lines.

Examples

```
LINE (10,10)-(100,100)
LINE (20,20)-(50,50),10
LINE (200,200)-(250,250),,B
LINE (220,220)-(230,230),12,BF
LINE -(300,300)
LINE -(100,100),5
LINE -(80,80),2,BF
```

Mode

Program only

See Also

BOX, **CHORD**, **CIRCLE**, **DRAW**, **GRAPH**, **PAINT**, **PCOLOR**, **PIE**

7.118 LINE EDIT Statement (Version 1.2 or later)

Edits a string.

LINE EDIT[**(***string***)**] [*prompt*{**;** | **,**}] *variable*

Parameters

string
String to be edited.

prompt
String displayed before data is entered. A semicolon after *prompt* appends a question mark (?) to the prompt string.

variable
Variable in which the data read from keyboard or file is stored.

Remarks

The **LINE EDIT** statement reads all characters up to a carriage return.

If string is omitted the default is an empty string.

Examples

```
LINE EDIT "Greeting"; A$
LINE EDIT(N$) "Name: ", N$
```

Mode

Program only

See Also

INPUT, **LINE INPUT**

7.119 LINE INPUT Statement

Reads input from the keyboard or a file.

LINE INPUT [*prompt*{**;** | **,**}] *variable*

- or -

LINE INPUT #*file***,***variable*

Parameters

prompt
String displayed before data is entered. A semicolon after *prompt* appends a question mark (?) to the prompt string.

variable
Variable in which the data read from keyboard or file is stored.

file
File number of open file.

Remarks

The **LINE INPUT** statement reads all characters up to a carriage return.

Examples

```
LINE INPUT A$
LINE INPUT "NAME"; N$
LINE INPUT "NAME: ", N$
LINE INPUT #1,A$
```

Mode

Program only

See Also

INPUT, **OPEN**

7.120 LINES Statement

Displays the number of lines in the current program.

LINES

Mode

Immediate only

7.121 LIST Statement

Displays lines in the current program.

LIST [*line*][**-**[*line*]]

- or -

LIST [*line*][**-**[*line*]][,[*line*][**-**[*line*]]]... (Version 1.2 or later)

Parameters

line
Line number of line to display.

Remarks

If *line* is omitted all lines are displayed.

Examples

```
LIST
LIST 100
LIST 10-
LIST 10-100
LIST -100
LIST -10,20,50-100,900-
```

Mode

Immediate only

7.122 LLIST Statement (Version 1.1 or later, standard version only)

Prints lines in the current program to the printer.

LLIST [*line*][**-**[*line*]]

Parameters

line
Line number of line to printed.

Remarks

If *line* is omitted all lines are printed.

Examples

```
LLIST
LLIST 100
LLIST 10-
LLIST 10-100
LLIST -100
```

Mode

Immediate only

7.123 LOAD Statement

Loads a program.

LOAD *filespec*[**,R**]

- or -

L. *filespec*[**,R**] (Version 1.5 or later)

Parameters

filespec
File name of program to load.

Remarks

The **LOAD** statement can be used to load a BASIC program. The new program replaces the current program. The program can be in NBASIC binary or ASCII format.

Programs in ASCII format that do not have line numbers may also be loaded. The lines are appended to the program as they occur in the file. If lines include line numbers, they are inserted in order and subsequent lines replace any duplicates.

Use **,R** to run the program immediately.

The default extension for the **LOAD** statement if not specified in the *filespec* is "nba".

Examples

```
LOAD "ANALYZE.BAS"
LOAD "PROGRAMS:COUNT.NBA",R
```

Mode

Immediate, Program

See Also

ASAVE, **BSAVE**, **SAVE**

7.124 LOADC Statement

Loads a program from the Windows clipboard.

LOADC[**,R**]

Remarks

The **LOADC** statement can be used to load a BASIC program from the Windows clipboard. The new program replaces the current program. The program on the clipboard must be ASCII text.

Programs in ASCII format that do not have line numbers may also be loaded. The lines are appended to the program as they occur in the file. If lines include line numbers, they are inserted in order and subsequent lines replace any duplicates.

Use **,R** to run the program immediately.

Examples

```
LOADC
LOADC,R
```

Mode

Immediate only

See Also

SAVEC

7.125 LOADR Statement

Loads a built-in program.

LOADR *program*[**,R**]

Parameters

program
Name of program to load.

Remarks

Use **,R** to run the program immediately.

Examples

```
LOADR "WELCOME"
LOADR "WELCOME",R
```

Mode

Immediate only

See Also

DIRR, **RUNR**

7.126 LOCATE Statement

Moves the cursor to a specified position on the screen.

LOCATE *row*,*column*

Parameters

row
Row where cursor is to be moved.

column
Column where cursor is to be moved.

Remarks

The screen has 25 rows numbered 0 through 24 and 80 columns numbered 0 through 79.

Examples

```
LOCATE 10,0
```

Mode

Immediate, Program

7.127 LOCK Statement

Locks the screen.

LOCK #0

Remarks

The **LOCK** statement locks the screen and does not display changes to the screen immediately. Use the **UNLOCK** statement to unlock the screen and display the changes.

Mode

Program only

See Also

UNLOCK

7.128 LOG Statement (standard version only)

Opens a file and begins logging screen output to the log file or turns logging on or off or stops logging and closes the log file.

LOG [**TO**] *filespec*[**,A**]

- or -

LOG {**ON** | **OFF** | **STOP**}

Parameters

filespec
Name of log file.

Remarks

The **LOG** statement can be used record output from a program to a file. Form 1 opens the log file and turns logging on. Logging can be turned off and on at any time. When logging is stopped, the log file is closed and must be opened again.

The default extension for the **LOG** statement if not specified in the *filespec* is "log".

Use **,A** to append to the log file.

Examples

```
LOG TO "OUTPUT.TXT"
LOG "OUTPUT.TXT",A
LOG ON
LOG OFF
LOG STOP
```

Mode

Immediate, Program

See Also

LOG$

7.129 LPRINT Statement (Version 1.1 or later, standard version only)

Writes output to the printer.

LPRINT [*expression*] [{**,** | **;** | **SPC(***spaces***)** | **TAB(***column***)**} [*expression*]]...

Parameters

expression
Data to print.

spaces
Number of spaces to output.

column

Column where output is to start.

Remarks

A **,** (comma) starts printing at the next print zone. Print zones are 10 characters wide. A **;** starts printing immediately after the last value printed. **SPC** prints a specified number of spaces. **TAB** starts printing at the specified column.

If any delimiters (**,** | **;** | **SPC** | **TAB**) are not specified as the last item in the print statement a carriage return is printed moving the print position to the beginning of the next line.

Examples

```
LPRINT
LPRINT "HELLO"
LPRINT "ABC","123"
LPRINT 100;" APPLES"
LPRINT "TOTAL:" SPC(5) T
LPRINT "ITEMS:";I
```

Mode

Program only

See Also

LPRINT USING, **OPEN PRINTER**, **LPOS**

7.130 LPRINT USING Statement (Version 1.1 or later, standard version only)

Writes output to the printer.

LPRINT USING *format*[**;**] *expression*[**,***expression*]...[**;**]

Parameters

format
Format string.

expression
Data to print.

Remarks

The format string consists of characters that specify how numbers and strings are formatted.

Numeric format characters:

#	Digit position.
.	Decimal point position.
,	Placed left of the decimal point, prints a comma in every third position.
+	Sign position.
-	Placed after digit, prints trailing sign for negative numbers.
^^^^	Prints in scientific notation (exponential) format.
$$	Prints leading $.
**	Fills leading spaces with *.
**$	Fills leading spaces with * and prints leading $.

String format characters:

&	Prints entire string.
!	Prints only the first character of a string.
\ \	Prints first n characters of a string where n is the number of spaces between the slashes + 2.

Other format characters:

_	Prints the following character as a literal.

If **;** is not specified as the last item in the print statement a carriage return is printed moving the print position to the beginning of the next line.

Examples

```
LPRINT USING "TOTAL: #####"; 1000
LPRINT USING "#####,.##"; 1000.21
LPRINT USING "####-"; -123
LPRINT USING "$$###.##"; 121.95
LPRINT USING "**###.##"; 121.95
LPRINT USING "**$##.##"; 121.95
LPRINT USING "**$####,.##-"; -1021.95
LPRINT USING "&"; "TEST STRING"
LPRINT USING "\  \"; "TEST STRING"
LPRINT USING "!"; "TEST STRING"
LPRINT USING "_###"; 5
```

Mode

Program only

See Also

LPRINT, **OPEN PRINTER**, **LPOS**

7.131 LSET Statement

Left justifies data in the field variable and moves the data into the file buffer or in a string variable.

LSET *variable*=*expression*

Parameters

variable
String variable to assign value to.

expression
Value to assign to variable.

Examples

```
LSET B$=S$
LSET N$="TEST"
```

Mode

Immediate, Program

See Also

FIELD, **RSET**

7.132 MERGE Statement

Merges a program with the current program.

MERGE *filespec*[**,***new*[**,***increment*]]

- or -

ME. *filespec*[**,***new*[**,***increment*]] (Version 1.5 or later)

Parameters

filespec
File name of program to merge.

new
Starting line number where program should be merged and renumbered.

increment
Increment for renumbering merged program.

Remarks

The **MERGE** statement can be used to merge a BASIC program with the current program. The program can be in NBASIC binary or ASCII format.

A line in the merged program replaces a line in the current program having the same line number.

The default extension for the **MERGE** statement if not specified in the *filespec* is "nba".

Examples

```
MERGE "HEADER"
MERGE "PROGRAMS:SUBS.BAS",1000
MERGE "DATA.BAS",5000,5
```

Mode

Immediate only

7.133 MID$ Statement

Assigns part of a string variable to another string.

MID$(*variable***,***start*[**,***length*]**)=***expression*

Parameters

variable
String variable to assign string to.

start

Starting index.

length
Length of substring.

expression
String to assign.

Remarks

The **MID$** statement replaces *length* characters in *variable* beginning at index *start* with characters from *expression*.

If *length* is omitted the default is 1.

Examples

```
MID$(A$,5,4)="TEST"
MID$(B$,1)="0"
```

Mode

Immediate, Program

See Also

LEFT$**, **RIGHT$

7.134 MOVE Statement

Moves a file to another file or volume.

MOVE *filespec1* **TO** *filespec2*

- or -

M. *filespec1* **TO** *filespec2* (Version 1.5 or later)

Parameters

filespec1
Name of existing file.

filespec2
Name of new file.

Remarks

The **MOVE** statement can be used to move an existing file to another file or volume.

The *filespec* and *filespec2* parameters may contain the wildcard characters * and ?. (Version 1.5 or later)

Examples

```
MOVE "TEST.DAT" TO "CONFIG.DAT"
MOVE "PROGRAMS:TEST.DAT" TO "DATA:TEST.DAT"
MOVE "*.DOC" TO "TEMP:"
MOVE "*.TXT" TO "*.DOC"
```

```
MOVE "*.LOG" TO "HISTORY.TXT"
```

Mode

Immediate, Program

See Also

APPEND, **COPY**, **RENAME**

7.135 MOVE Statement, editing (standard version only)

Moves a line in the current program to another line.

MOVE *line1* **TO** *line2*[**,E**]

- or -

M. *line1* **TO** *line2*[**,E**] (Version 1.5 or later)

Parameters

line1
Line number of line to move.

line2
Line number of destination line.

Remarks

The **MOVE** statement moves an existing line in the current program to another line. The destination line is overwritten if it already exists. Line numbers in the current program that refer to the line being moved are automatically changed to the destination line number.

Use **,E** to edit the destination line.

Examples

```
MOVE 10 TO 50
MOVE 20 TO 100,E
```

Mode

Immediate only

See Also

APPEND, **COPY**, **SWAP**

7.136 NAME Statement

Renames a file.

NAME *filespec1* **AS** *filespec2*

Parameters

filespec1

Name of existing file.

filespec2
New name of file.

Remarks

The **NAME** statement cannot change the volume where the file is stored, only the name and extension of the file can be changed.

The **NAME** statement is equivalent to the **RENAME** statement and is provided for compatibility.

Examples

```
NAME "TEST.DAT" AS "CONFIG.DAT"
NAME "PROGRAMS:TEST.DAT" AS "PROGRAMS:INDEX.DAT"
```

Mode

Immediate, Program

See Also

RENAME

7.137 NEW Statement

Erases the current program.

NEW

Mode

Immediate, Program

See Also

INIT

7.138 NEXT Statement

Increments and tests the variable in a FOR loop.

NEXT [*counter*[**,***counter*]...]

Parameters

counter
Numeric variable used as loop counter.

Remarks

The **NEXT** statement is used to increment and test the variable in a **FOR** loop and exit the loop if the counter has reached the end value.

If *counter* is omitted the *counter* in the most recent **FOR** loop is incremented. If multiple *counters* are specified place the counter of the most recent **FOR** loop first followed by the next most recent.

Examples

```
FOR I=1 TO 10: NEXT
FOR I=1 TO 10: NEXT I
FOR I=1 TO 10: FOR J=I TO 10: NEXT J,I 'Equivalent to NEXT J:
NEXT I
```

Mode

Immediate, Program

See Also

FOR

7.139 NOBREAK Statement (standard version only)

Clears breakpoints.

NOBREAK

- or -

NOBREAK *line*[**,***line*]...

- or -

NOBREAK [*line*][**-**[*line*]][,[*line*][**-**[*line*]]]... (Version 1.2 or later)

Parameters

line
Line number or label where breakpoint is to be cleared.

Remarks

The **NOBREAK** statement can be used to clear breakpoints set by **BREAK**.

NOBREAK with no parameters clears all breakpoints.

Examples

```
NOBREAK
NOBREAK 100,150
NOBREAK 200,*REDO
NOBREAK -10,20,50-100,900-
```

Mode

Immediate, Program

See Also

BREAK

7.140 NOINVERSE Statement (Version 1.5 or later)

Turns inverse off.

NOINVERSE

Mode

Immediate, Program

See Also

INVERSE

7.141 NOREVERSE Statement (Version 1.5 or later)

Turns inverse off.

NOREVERSE

Mode

Immediate, Program

See Also

INVERSE

7.142 NORMAL Statement (Version 1.4 or later)

Turns inverse off.

NORMAL

Mode

Immediate, Program

See Also

INVERSE

7.143 NOTRACE Statement (Version 1.4 or later)

Disables tracing of program.

NOTRACE

Mode

Immediate, Program

Remarks

The **NOTRACE** statement is equivalent to the **TROFF** statement and is provided for compatibility.

See Also

TRACE, **TROFF**

7.144 NUMBER Statement (Version 1.4 or later, standard version only)

Turns automatic line numbering on.

NUM[**BER**] [*initial*[**,***increment*]]...

Parameters

initial
Starting line number.

increment
Increment for each new line number.

Remarks

The **NUMBER** statement can be used to automatically generate line numbers when entering a program. After each line is entered, the line number is incremented by increment and is automatically entered as part of the next line. If a line already exists with this line number, a new line number is not automatically generated.

If initial is omitted the default is 100. If increment is omitted the default is 10.

The **NUMBER** statement is similar to the **AUTO** statement and is provided for compatibility.

Examples

```
NUMBER
NUMBER 1000
NUMBER 10,5
NUM
NUM 1000
NUM 10,5
```

Mode

Immediate only

See Also

AUTO

7.145 OLD Statement

Loads a program or restores a program.

OLD *filespec*

- or -

OLD (Version 1.3 or later)

Parameters

filespec
File name of program to load.

Remarks

The **OLD** statement can be used to load a BASIC program. The new program replaces the current program. The program can be in NBASIC binary or ASCII format.

Programs in ASCII format that do not have line numbers may also be loaded. The lines are appended to the program as they occur in the file. If lines include line numbers, they are inserted in order and subsequent lines replace any duplicates.

The default extension for the **OLD** statement if not specified in the *filespec* is "nba".

The **OLD** statement can also be used to restore a program erased by the **NEW** statement. A program cannot be restored if the current program has been modified.

Examples

```
OLD "ANALYZE.BAS"
OLD "PROGRAMS:COUNT.NBA"
OLD
```

Mode

Immediate, Program

See Also

LOAD, **NEW**

7.146 ON BREAK Statement

Enables BREAK key trapping and when the BREAK key is pressed transfers control to a BREAK key handling routine or suspends BREAK key trapping.

ON BREAK GOTO *line*

- or -

ON BREAK GOTO 0

Parameters

line
Line number or label of first line of handling routine.

Remarks

If BREAK key trapping is enabled, the **ON BREAK** statement branches to a subroutine whenever the BREAK key is pressed.

Use the **RESUME** statement to return from the subroutine.

Examples

```
ON BREAK GOTO 1000
```

```
ON BREAK GOTO *HANDLER
ON BREAK GOTO 0
```

Mode

Program only

See Also

BREAK, **RESUME**

7.147 ON ERROR Statement

Enables error handling and when a run-time error occurs transfers control to an error handling routine or resumes execution or suspends error handling.

ON ERROR GOTO *line*

- or -

ON ERROR RESUME NEXT

- or -

ON ERROR GOTO 0

Parameters

line
Line number or label of first line of handling routine.

Remarks

The **ON ERROR RESUME NEXT** statement does not branch to a handling subroutine, it resumes execution with the next statement following the statement that caused the error.

Use the **RESUME** statement to return from the subroutine.

Examples

```
ON ERROR GOTO 1000
ON ERROR GOTO *HANDLER
ON ERROR RESUME NEXT
ON BREAK GOTO 0
```

Mode

Program only

See Also

ERL, **ERR**, **ERR$**, **RESUME**

7.148 ON GOSUB Statement

Transfers control of the current program to one of several subroutines based on the value of an expression.

ON *expression* **GOSUB** *line*[,*line*]...

Parameters

expression
Expression that evaluates to a number in the range 0 to 255.

line
Line number or label of subroutine.

Remarks

If the value of *expression* is 1, the program branches to the first line specified; if *expression* is 2, it branches to the second line, and so on.

Use the **RETURN** statement to return from the subroutine.

Examples

```
ON INDEX GOSUB 100,200
```

Mode

Immediate, Program

See Also

IF, **GOSUB**, **ON GOTO**, **RETURN**

7.149 ON GOTO Statement

Transfers control of the current program to one of several lines based on the value of an expression.

ON *expression* **GOTO** *line*[,*line*]...

Parameters

expression
Expression that evaluates to a number in the range 0 to 255.

line
Line number or label of line.

Remarks

If the value of *expression* is 1, the program branches to the first line specified; if *expression* is 2, it branches to the second line, and so on.

Examples

```
ON INDEX GOTO 100,200
```

Mode

Immediate, Program

See Also

IF, **GOTO**, **ON GOSUB**

7.150 ON TIMER Statement

Enables timer event trapping and when a timer event occurs transfers control to a timer event handling routine or suspends timer event trapping.

ON TIMER[**(***interval***)**] **GOTO** *line*

- or -

ON TIMER GOTO 0

Parameters

interval
Timer event interval in milliseconds.

line
Line number or label of first line of handling routine.

Remarks

If timer event trapping is enabled, the **ON TIMER** statement branches to a subroutine whenever the specified number of milliseconds has elapsed.

Use the **RESUME** statement to return from the subroutine.

If *interval* is omitted the default is 1000 milliseconds (1 second).

Examples

```
ON TIMER GOTO 1000
ON TIMER(5000) GOTO *HANDLER
ON TIMER GOTO 0
```

Mode

Program only

See Also

GOTO TIMER, **RESUME**, **TIMER**

7.151 OPEN Statement

Opens a file.

OPEN *filespec* **FOR** {**APPEND** | **INPUT** | **OUTPUT** | **RANDOM**} **AS** *#file* [**LEN=***length*]

- or -

OPEN *mode*,*#file*,*filespec*[,*length*]

Parameters

filespec
Name of file to open.

file

File number that identifies open file.

length
For random access files, record length; for sequential files, characters buffered.

mode
File mode.

Remarks

APPEND opens the file for sequential output and positions the file pointer to the end of the file. Output to the file extends (appends to) the file.

INPUT opens the file for sequential input.

OUTPUT opens the file for sequential output.

RANDOM opens the file for random access.

If *length* is omitted the default is 128 for random access files and 512 for sequential files.

The default extension for the **OPEN** statement if not specified in the *filespec* is "dat".

The file mode is a string specifying the file open mode; "A" for append, "I" for input, "O" for output, or "R" for random.

Examples

```
OPEN "CONFIG.DAT" FOR INPUT AS #1
OPEN "TEST.DAT" FOR OUTPUT AS #2 LEN=100
OPEN "I",#1,"CONFIG.DAT"
OPEN "O",#2,"TEST.DAT",100
```

Mode

Immediate, Program

See Also

CLOSE, **DELETE**, **FIELD**, **GET**, **INPUT**, **INSERT**, **LINE INPUT**, **PUT**, **REOPEN**, **RESET**, **REWIND**, **TRUNCATE**, **UNLOAD**, **WRITE**

7.152 OPEN PRINTER Statement (Version 1.1 or later, standard version only)

Opens a printer.

OPEN PRINTER

Remarks

The **OPEN PRINTER** statement opens the most recent printer set by using the **PRINTER** or **PRINTER?** statements or if not set, the current printer as specified in the Print Setup or Print dialogs and starts a print job.

Mode

Program only

See Also

CLOSE PRINTER, **LPRINT**, **LPRINT USING**, **PRINTER**, **PRINTER?**, **LPOS**

7.153 OPTION BASE Statement (Version 1.2 or later)

Specifies the base index of arrays.

OPEN BASE {**0** | **1**}

Remarks

If **OPTION BASE 1** is specified in a program, the first element in an array dimension is in location 1. If no base is set or **OPTION BASE 0** is specified, the first element in an array dimension is in location 0.

The **OPTION BASE** statement must be specified before any arrays are created in a program.

Examples

```
OPTION BASE 0
OPTION BASE 1
```

Mode

Program only

See Also

DIM, OPTION EXPLICIT, REDIM, LBOUND, MAXSIZE, SIZE, UBOUND

7.154 OPTION EXPLICIT Statement (Version 1.2 or later)

Specifies that arrays must be explicitly created.

OPEN EXPLICIT

Remarks

If **OPTION EXPLICIT** is specified in a program, all arrays must be explicitly created using the **DIM** or **REDIM** statements.

The **OPTION EXPLICIT** statement must be specified before any arrays are created in a program.

Mode

Program only

See Also

DIM, OPTION BASE, REDIM

7.155 PAINT Statement (standard version only)

Fills a graphics area with a color.

PAINT (*x***,***y***)[,**[*color*][**,***border*]]

Parameters

x
X coordinate where painting begins.

y
Y coordinate where painting begins.

color
Color to paint.

border
Color of border where painting stops.

Remarks

If *color* is omitted the default is the current graphics foreground color. If *border* is omitted the default is the current graphics background color.

Graphics must be turned on to paint a graphics area.

Examples

```
PAINT (100,100)
PAINT (50,50),7
PAINT (200,50),5,7
```

Mode

Program only

See Also
BOX, **CHORD**, **CIRCLE**, **DRAW**, **GRAPH**, **LINE**, **PIE**, **PCOLOR**

7.156 PAUSE Statement

Pauses a program.

PAUSE [*duration*]

Parameters

duration
Number of milliseconds to pause.

Remarks

The **PAUSE** statement pauses a program for the duration specified or until a key is pressed.

If *duration* is omitted the default is 1000 milliseconds.

Examples

```
PAUSE
PAUSE 2000
```

Mode

Immediate, Program

See Also

WAIT

7.157 PCLR Statement (standard version only)

Clears a rectangular area of the graphics screen.

PCLR (*x1*,*y1***)-(***x2*,*y2***)**[**,***color*]

Parameters

x1
X coordinate of corner of rectangle.

y1
Y coordinate of corner of rectangle.

x2
X coordinate of opposite corner of rectangle.

y2
Y coordinate of opposite corner of rectangle.

color
Color to clear rectangle with.

Remarks

The **PCLR** statement clears a rectangular area of the graphics screen with *color* and sets the graphics background color to *color*.

If *color* is omitted the default is the current graphics background color.

Graphics must be turned on to clear the graphics screen.

Examples

```
PCLR (100,100)-(200,200)
PCLR (100,100)-(200,200),1
```

Mode

Program only

See Also

PCLS, **PCOLOR**

7.158 PCLS Statement (standard version only)

Clears the graphics screen.

PCLS [*color*]

Parameters

color
Color to clear screen with.

Remarks

The **PCLS** statement clears the graphics screen with *color* and sets the graphics background color to *color*.

If *color* is omitted the default is the current graphics background color.

Graphics must be turned on to clear the graphics screen.

Examples

```
PCLS
PCLS 7
```

Mode

Program only

See Also

PCLR, **PCOLOR**

7.159 PCOLOR Statement (standard version only)

Sets the graphics colors.

PCOLOR *foreground*[**,***background*]

Parameters

foreground
Foreground graphics color.

background
Background graphics color.

Remarks

The **PCOLOR** statement can be used to set the foreground and background graphics colors.

If *background* is omitted the default is the current graphics background color.

Graphics must be turned on to set the graphics colors.

Examples

```
PCOLOR 4
PCOLOR 7,10
```

Mode

Program only

See Also

PCLR, **PCLS**

7.160 PCP Statement (Version 1.2 or later, standard version only)

Displays the current printer.

PCP

Mode

Immediate only

See Also

PRINTER

7.161 PFONT Statement (standard version only)

Changes the graphics font size.

PFONT *size*

Parameters

size
Size of graphics font.

Remarks

Graphics must be turned on to set the graphics font size.

Examples

```
PFONT 8
```

Mode

Program only

See Also

PPRINT

7.162 PIE Statement (standard version only)

Draws a pie.

PIE (*x***,***y***),***radius*[,[*color*][,[*start*][,[*end*][,[*aspect*][,**F**]]]]]

Parameters

x
X coordinate of center of pie.

y
Y coordinate of center of pie.

radius
Radius of pie.

color
Color of pie.

start
Starting angle of pie in radians.

end
Ending angle of pie in radians.

aspect
Ratio of length of y-axis to length of x-axis used to draw elliptical pies.

Remarks

The **PIE** statement can be used to draw a pie of a circle or ellipse. A pie is an arc specified by the *start* and *end* angles with the endpoints connected to the center point.

Use **,F** to fill in the pie.

If *color* is omitted the default is the current graphics foreground color. If *start* is omitted the default is 0. If *end* is omitted the default is 0. if *aspect* is omitted the default is 1.

Graphics must be turned on to draw pies.

Examples

```
PIE (50,50),25,,.785,3.142
PIE (100,100),50,5,,2.356,,F
```

Mode

Program only

See Also

BOX, **CHORD**, **CIRCLE**, **DRAW**, **GRAPH**, **LINE**, **PAINT**, **PCOLOR**

7.163 PLOT Statement (Version 1.4 or later, standard version only)

Draws a point.

PLOT *x*,*y*

Parameters

x

X coordinate of point.

y
Y coordinate of point.

Remarks

The **PLOT** statement draws a point at the specified coordinates using the current graphics foreground color.

The **PLOT** statement is similar to the **PSET** statement and is provided for compatibility.

Graphics must be turned on to draw points.

Examples

```
PLOT 50,50
```

Mode

Program only

See Also

COLOR, **GR**, **HPLOT**, **PSET**

7.164 POKE Statement

Writes a byte value to a memory location.

POKE *address*,*value*

Parameters

address
Address of memory location.

value
Value to write to memory location.

Examples

```
POKE 1994,5
POKE 1994,255
```

Mode

Immediate, Program

See Also

EXEC, **PEEK**

7.165 POP Statement

Removes a return address from the call stack.

POP

Remarks

The **POP** statement removes the most recent return address placed on the call stack by a **GOSUB** statement.

Mode

Immediate, Program

See Also

GOSUB, **PUSH**, **RETURN**

7.166 PPRINT Statement (standard version only)

Writes output to the graphics screen.

PPRINT (*x***,***y***)[,**[*color*][,[*mode*][,[*angle*][,[*horizontal*][,*vertical*]]]]]**;** [*expression*] [{**,** | **;** | **SPC(***spaces***)**} [*expression*]]...

Parameters

x
X coordinate where output is to begin.

y
Y coordinate where output is to begin.

color
Text color.

mode
Background mode, 0 for opaque or 1 for transparent.

angle
Text angle, 0 - 0 degrees, 1 - 90 degrees, 2 - 180 degrees, 3 - 270 degrees.

horizontal
Horizontal alignment, 0 - left, 1 - center, 2 - right.

vertical
Vertical alignment, 0 - top, 1 - center, 2 - bottom.

expression
Data to print.

spaces
Number of spaces to output.

Remarks

A **,** (comma) or a **;** starts printing immediately after the last value printed. **SPC** prints a specified number of spaces.

If *color* is omitted the default is the current graphics foreground color. If *mode* is omitted the default is 0 (opaque). If *angle* is omitted the default is 0. If *horizontal* is omitted the default is 0 (left). If *vertical* is omitted the default is 0 (top).

PPRINT does not print a carriage return.

Graphics must be turned on to print to the graphics screen.

Examples

```
PPRINT (10,10),5; "COUNT"
PPRINT (10,25),5,,3; "ITEM"
```

Mode

Program only

See Also

PPRINT USING

7.167 PPRINT USING Statement (standard version only)

Writes output to the graphics screen.

PPRINT (*x***,***y***)[,**[*color*][,[*mode*][,[*angle*][,[*horizontal*][,*vertical*]]]]]**; USING** *format*; *expression*[**,***expression*]...

Parameters

x
X coordinate where output is to begin.

y
Y coordinate where output is to begin.

color
Text color.

mode
Background mode, 0 for opaque or 1 for transparent.

angle
Text angle, 0 - 0 degrees, 1 - 90 degrees, 2 - 180 degrees, 3 - 270 degrees.

horizontal
Horizontal alignment, 0 - left, 1 - center, 2 - right.

vertical
Vertical alignment, 0 - top, 1 - center, 2 - bottom.

format
Format string.

expression
Data to print.

Remarks

If *color* is omitted the default is the current graphics foreground color. If *mode* is omitted the default is 0 (opaque). If *angle* is omitted the default is 0. If *horizontal* is omitted the default is 0 (left). If *vertical* is omitted the default is 0 (top).

The format string consists of characters that specify how numbers and strings are formatted.

Numeric format characters:

#	Digit position.
.	Decimal point position.
,	Placed left of the decimal point, prints a comma in every third position.
+	Sign position.
-	Placed after digit, prints trailing sign for negative numbers.
^^^^	Prints in scientific notation (exponential) format.
$$	Prints leading $.
**	Fills leading spaces with *.
**$	Fills leading spaces with * and prints leading $.

String format characters:

&	Prints entire string.
!	Prints only the first character of a string.
\ \	Prints first n characters of a string where n is the number of spaces between the slashes + 2.

Other format characters:

_	Prints the following character as a literal.

PPRINT USING does not print a carriage return.

Graphics must be turned on to print to the graphics screen.

Examples

```
PPRINT (100,10),5; USING "$$###.##"; 121.25
```

Mode

Program only

See Also

PPRINT

7.168 PRESET Statement (standard version only)

Draws a point.

PRESET (*x***,***y***)**[**,***color*]

Parameters

x
X coordinate of point.

y
Y coordinate of point.

color
Color of point.

Remarks

If *color* is omitted the default is the current graphics background color.

Graphics must be turned on to draw points.

Examples

```
PRESET (50,50)
PRESET (100,100),5
```

Mode

Program only

See Also

BOX, **GRAPH**, **LINE**, **PAINT**, **PCOLOR**, **POINT**, **PSET**

7.169 PRINT Statement

Writes output to the screen, a file or the printer.

{**PRINT** | **?**} [*#file***,**] [*expression*] [{**,** | **;** | **SPC(***spaces***)** | **TAB(***column***)**} [*expression*]]...

Parameters

file
File number of open file or printer (-2).

expression
Data to print.

spaces
Number of spaces to output.

column
Column where output is to start.

Remarks

A **,** (comma) starts printing at the next print zone. Print zones are 10 characters wide. A **;** starts printing immediately after the last value printed. **SPC** prints a specified number of spaces. **TAB** starts printing at the specified column.

If any delimiters (**,** | **;** | **SPC** | **TAB**) are not specified as the last item in the print statement a carriage return is printed erasing the rest of the line and moving the print position to the beginning of the next line.

Use **PRINT #-2** to write output to the printer.

Examples

```
PRINT
PRINT "HELLO"
PRINT "ABC","123"
PRINT 100;" APPLES"
PRINT "TOTAL:" SPC(5) T
PRINT #1, "ITEMS:";I
```

Mode

Immediate, Program

See Also

OPEN, **PRINT USING**, **WRITE**

7.170 PRINT USING Statement

Writes output to the screen, a file or the printer.

{**PRINT** | **?**} [*#file*,] **USING** *format*; *expression*[,*expression*]...[;]

Parameters

file
File number of open file or printer (-2).

format
Format string.

expression
Data to print.

Remarks

The format string consists of characters that specify how numbers and strings are formatted.

Numeric format characters:

#	Digit position.
.	Decimal point position.
,	Placed left of the decimal point, prints a comma in every third position.
+	Sign position.
-	Placed after digit, prints trailing sign for negative numbers.
^^^^	Prints in scientific notation (exponential) format.
$$	Prints leading $.
**	Fills leading spaces with *.
**$	Fills leading spaces with * and prints leading $.

String format characters:

&	Prints entire string.
!	Prints only the first character of a string.
\ \	Prints first n characters of a string where n is the number of spaces between the slashes + 2.

Other format characters:

_	Prints the following character as a literal.

If **;** is not specified as the last item in the print statement a carriage return is printed erasing the rest of the line and moving the print position to the beginning of the next line.

Use **PRINT #-2** to write output to the printer.

Examples

```
PRINT USING "TOTAL: #####"; 1000
```

```
PRINT USING "#####,.##"; 1000.21
PRINT USING "####-"; -123
PRINT USING "$$###.##"; 121.95
PRINT USING "**###.##"; 121.95
PRINT USING "**$##.##"; 121.95
PRINT USING "**$####,.##-"; -1021.95
PRINT USING "&"; "TEST STRING"
PRINT USING "\  \"; "TEST STRING"
PRINT USING "!"; "TEST STRING"
PRINT USING "_###"; 5
```

Mode

Immediate, Program

See Also

OPEN, **PRINT**, **WRITE**

7.171 PRINTER Statement (Version 1.1 or later, standard version only)

Sets the current printer.

PRINTER=*printer*

Parameters

printer
Name of the printer to set as the current printer.

Remarks

The **PRINTER** statement sets the current printer used by the **LLIST** and **OPEN PRINTER** statements. The name of the printer must match a printer installed on the host operating system.

Mode

Immediate, Program

See Also

OPEN PRINTER, **PRINTER?**

7.172 PRINTER? Statement (Version 1.1 or later, standard version only)

Sets the current printer.

PRINTER?

Remarks

The **PRINTER?** statement sets the current printer by selecting the printer from a list of available printers installed on the host operating system.

Mode

Immediate, Program

See Also

OPEN PRINTER, **PRINTER**

7.173 PRINTERS Statement (Version 1.2 or later, standard version only)

Lists available printers.

PRINTERS

Remarks

The **PRINTERS** statement lists the available printers installed on the host operating system.

Mode

Immediate only

See Also

OPEN PRINTER, **PRINTER**

7.174 PROFILE Statement (standard version only)

Displays the execution profile of the current program or turns profiling on or off.

PROFILE [*line*][**-**[*line*]]

- or -

PROFILE [*line*][**-**[*line*]][,[*line*][**-**[*line*]]]... (Version 1.2 or later)

- or -

PROFILE {**ON** | **OFF**}

Parameters

line
Line number of line whose profile will be displayed.

Remarks

If *line* is omitted the profiles for all lines are displayed.

Examples

```
PROFILE
PROFILE 100
PROFILE 10-
PROFILE 10-100
PROFILE -100
PROFILE ON
PROFILE OFF
```

Mode

Immediate only (first form), Immediate or Program (second form)

7.175 PROMPT Statement (standard version only)

Changes the prompt.

PROMPT *prompt*

Parameters

prompt
New prompt string.

Remarks

The **PROMPT** statement can be used to change the prompt displayed by the editor. Use "" as the prompt string to return to the default prompt.

Examples

```
PROMPT "OK"
PROMPT ""
```

Mode

Immediate, Program

7.176 PSET Statement (standard version only)

Draws a point.

PSET (*x***,***y***)[,***color***]**

Parameters

x
X coordinate of point.

y
Y coordinate of point.

color
Color of point.

Remarks

If *color* is omitted the default is the current graphics foreground color.

Graphics must be turned on to draw points.

Examples

```
PSET (50,50)
PSET (100,100),5
```

Mode

Program only

See Also

BOX, **GRAPH**, **LINE**, **PAINT**, **PCOLOR**, **POINT**, **PRESET**

7.177 PUSH Statement

Places a return address on the call stack.

PUSH [*line*]

Parameters

line
Line number or label of return point.

Remarks

The **PUSH** statement places the return address of the location specified on the call stack for a **RETURN** statement.

If *line* is omitted the address of the statement immediately following the **PUSH** statement is used (if there is no statement after PUSH, the return address points to the end of the line).

Examples

```
PUSH
PUSH 100
PUSH *START
```

Mode

Immediate, Program

See Also

GOSUB, **POP**, **RETURN**

7.178 PUT Statement

Writes a random access buffer to a file.

PUT #*file*,*record*

Parameters

file
File number of open random access file.

record
Record number of record to write.

Example

```
PUT #1,1
```

Mode

Immediate, Program

See Also

FIELD, **GET**, **LSET**, **OPEN**, **RSET**

7.179 PUT Statement, graphics (standard version only)

Displays a graphics screen image.

PUT (*x1***,***y1***)-(***x2***,***y2***),***variable*[,{**AND** | **OR** | **PSET** | **PRESET** | **XOR**}]

Parameters

x1
X coordinate of corner of graphics image.

y1
Y coordinate of corner of graphics image.

x2
X coordinate of opposite corner of graphics image.

y2
Y coordinate of opposite corner of graphics image.

variable
Array variable where image is stored.

Remarks

Each pixel in the image requires one element of array storage. The size of the array can be found by using the equation (*x2*-*x1*+1) * (*y2*-*y1*+1), where (*x1*,*y1*) is the upper left corner of the image and (*x2*,*y2*) is the lower right corner.

AND mergers the stored image with the graphics screen.
OR superimposes the stored image on the graphics screen.
PSET draws stored image erasing the graphics screen.
PRESET draws stored image in reverse colors erasing the graphics screen.
XOR draws a stored image or erases a previously drawn image while preserving the background.

Graphics must be turned on to display images.

Examples

```
PUT (110,110)-(120,120),IMAGE1
PUT (210,210)-(220,220),IMAGE1,PSET
```

Mode

Program only

See Also

GET, **GRAPH**

7.180 QUIT Statement

Exits NBASIC.

QUIT

Remarks

The **QUIT** statement exits NBASIC and prompts to save the current program if modified.

The **QUIT** statement is equivalent to the **EXIT** statement and is provided for compatibility.

Mode

Immediate only

See Also

EXIT

7.181 RANDOMIZE Statement

Initializes the random number generator.

RANDOMIZE

Mode

Immediate, Program

7.182 READ Statement

Reads values from a **DATA** statement and assigns them to variables.

READ *variable*[,*variable*]...

Parameters

variable
Variable to which data values are assigned.

Examples

```
READ ITEM
READ USER$,AGE
```

Mode

Immediate, Program

See Also

DATA, **RESTORE**

7.183 REDIM Statement

Declares an array.

REDIM [**PRESERVE**] *array***(***subscripts***)**[**,***array***(***subscripts***)**]...

Parameters

array
Name of array.

subscripts
Dimensions of array.

Remarks

The **REDIM** statement cannot change the number of dimensions only the size of each dimension. The size of each dimension can be increased or decreased.

If **PRESERVE** is specified, the existing elements of the array are not initialized.

An array may have up to ten dimensions and 65536 elements per dimension (including element 0). However, the maximum size of the array including all dimensions cannot exceed 16777216 elements. The maximum size of the array is also constrained by available memory.

Elements of numeric arrays are initialized to 0 and elements of string arrays are initialized to "".

If **OPTION BASE 1** is specified, array elements begin at index 1 not 0.

Examples

```
REDIM PRESERVE A(50,50)
REDIM B$(10),C(10,20)
```

Mode

Immediate, Program

See Also

DIM, **ERASE**, **OPTION BASE**, **OPTION EXPLICIT**

7.184 REM Statement

Allows remarks to be inserted into a program.

REM [*remark*]

- or -

' [*remark*]

Parameters

remark
Text of remark.

Remarks

Any text after **REM** or **'** is ignored when the program runs.

A remark may be placed after a statement if it is preceded by the single-quote (') form of **REM** or if **REM** is preceded by a colon (:).

Examples

```
REM This is a comment
' This is also a comment
PRINT "TEST": REM This is a comment after a print statement
PRINT "TEST" ' This is also a comment after a print statement
```

Mode

Immediate, Program

7.185 REMARK Statement (standard version only)

Remarks (comments) lines in the current program.

REMARK {*line*[**-***line*] | *-line*}

- or -

REMARK {*line*[**-***line*] | *-line*}[,{*line*[**-***line*] | *-line*}]... (Version 1.2 or later)

Parameters

line
Line number of line to remark.

Remarks

The **REMARK** statement adds the **REM** statement to the beginning of the specified lines, which makes the lines comments and prevents them from being executed.

Examples

```
REMARK 100
REMARK 10-90
REMARK -200
REMARK -10,20,50-100,900-
```

Mode

Immediate only

See Also

UNREMARK

7.186 RENAME Statement

Renames a file.

RENAME *filespec1* **TO** *filespec2*

- or -

RENAME *filespec1* {**AS** | **TO**} *filespec2* (Version 1.4 or later)

- or -

R. *filespec1* {**AS** | **TO**} *filespec2* (Version 1.5 or later)

Parameters

filespec1
Name of existing file.

filespec2
New name of file.

Remarks

The **RENAME** statement cannot change the volume where the file is stored, only the name and extension of the file can be changed.

The *filespec* and *filespec2* parameters may contain the wildcard characters * and ?. (Version 1.5 or later)

Examples

```
RENAME "TEST.DAT" TO "CONFIG.DAT"
RENAME "PROGRAMS:TEST.DAT" TO "PROGRAMS:INDEX.DAT"
RENAME "*.TXT" TO "*.DOC"
```

Mode

Immediate, Program

See Also

APPEND, **COPY**, **MOVE**

7.187 RENUM Statement

Renumbers lines in the current program.

RENUM [*new*[,[*start*][,*increment*]]]

Parameters

new
New line number of first line that is renumbered.

start
Line number of line where renumbering is to start.

increment
Increment for next new line number.

Remarks

If *new* is omitted the default is 10. If *start* is omitted the default is the first line in the current program. If *increment* is omitted the default is 10.

Examples

```
RENUM
RENUM 10
RENUM 100,50
RENUM 100,50,1
RENUM 10,,5
```

Mode

Immediate only

See Also

UNNUM

7.188 RENUMBER Statement (Version 1.4 or later)

Renumbers lines in the current program.

RENUMBER [*new*[,[*start*][,*increment*]]]

Parameters

new
New line number of first line that is renumbered.

start
Line number of line where renumbering is to start.

increment
Increment for next new line number.

Remarks

If *new* is omitted the default is 10. If *start* is omitted the default is the first line in the current program. If *increment* is omitted the default is 10.

The **RENUMBER** statement is equivalent to the **RENUM** statement and is provided for compatibility.

Examples

```
RENUMBER
RENUMBER 10
RENUMBER 100,50
RENUMBER 100,50,1
RENUMBER 10,,5
```

Mode

Immediate only

See Also

RENUM

7.189 REOPEN Statement

Reopens a file in a different mode.

REOPEN #*file* **FOR** {**APPEND** | **INPUT** | **OUTPUT** | **RANDOM**}

- or -

REOPEN *mode*,**#***file*

Parameters

file
File number that identifies open file.

mode
File mode.

Remarks

The **REOPEN** statement closes the file and then reopens the file with the same file name, buffer length, and file number as specified in the **OPEN** statement that originally opened the file.

APPEND opens the file for sequential output and positions the file pointer to the end of the file. Output to the file extends (appends to) the file.

INPUT opens the file for sequential input.

OUTPUT opens the file for sequential output.

RANDOM opens the file for random access.

The file mode is a string specifying the file open mode; "A" for append, "I" for input, "O" for output, or "R" for random.

Examples

```
REOPEN #1 FOR INPUT
REOPEN "I",#1
```

Mode

Immediate, Program

See Also

OPEN

7.190 RESEQUENCE Statement (Version 1.4 or later)

Renumbers lines in the current program.

RES[**EQUENCE**] [[*initial*][,*increment*]]]

Parameters

initial

Line number of line where renumbering is to start.

increment
Increment for next new line number.

Remarks

If initial is omitted the default is 100. If increment is omitted the default is 10.

The **RESEQUENCE** statement is similar to the **RENUM** statement and is provided for compatibility.

Examples

```
RESEQUENCE
RESEQUENCE 10
RESEQUENCE 100,50
RESEQUENCE ,50
RES
RES 10
RES 100,50
RES ,1
```

Mode

Immediate only

See Also

RENUM

7.191 RESET Statement

Closes all open files.

RESET

Mode

Immediate, Program

See Also

CLOSE, **OPEN**

7.192 RESTORE Statement

Allows READ to reread values specified in **DATA** statements.

RESTORE [*line*]

Parameters

line
Line number or label of line containing data.

Remarks

If *line* is omitted the **READ** statement reads data from the first line in the current program that contains a **DATA** statement.

Examples

```
RESTORE
RESTORE 10
```

Mode

Immediate, Program

See Also

DATA, **READ**

7.193 RESUME Statement

Resumes program execution after a break, error, or timer handling routine.

RESUME [**0**]

- or -

RESUME *line*

- or -

RESUME NEXT

Parameters

line
Line number or label of line where execution is to resume.

Remarks

The **RESUME** [**0**] statement resumes execution with the statement that was interrupted by the handling routine. The **RESUME** *line* statement resumes execution with the specified line. The **RESUME NEXT** statement resumes execution with the next statement following the statement that was interrupted by the handling routine.

Examples

```
RESUME
RESUME 0
RESUME 100
RESUME *START
RESUME NEXT
```

Mode

Program only

See Also

ON BREAK, **ON ERROR**, **ON TIMER**

7.194 RETURN Statement

Returns from a subroutine.

RETURN

- or -

RETURN *line* (Version 1.5 or later)

Parameters

line
Line number or label of line to return to.

Examples

```
RETURN
RETURN 100
RETURN *START
```

Mode

Immediate, Program

See Also

GOSUB, **ON GOSUB**, **POP**, **PUSH**

7.195 REVERSE Statement (Version 1.5 or later)

Turns inverse on or off.

REVERSE {[**ON**] | **OFF**}

Remarks

The **REVERSE** statement reverses the foreground and background colors.

The **REVERSE** statement is equivalent to the **INVERSE** statement and is provided for compatibility.

Examples

```
REVERSE
REVERSE ON
REVERSE OFF
```

Mode

Immediate, Program

See Also

INVERSE

7.196 REWIND Statement

Rewinds one or more open files.

REWIND

- or -

REWIND #*file*[**,**#*file*]...

Parameters

file
File number of open file to rewind.

Remarks

The **REWIND** statement writes any data in the file buffer to the file and rewinds the file to the beginning. Input from files opened for input access will begin reading from the beginning of the file and output to files opened for output or append access will begin writing starting at the beginning of the file, overwriting existing file contents. The end of file for input access files may be set if the file is empty.

REWIND with no parameters rewinds all open files.

Examples

```
REWIND
REWIND #1
```

Mode

Immediate, Program

See Also

OPEN

7.197 RIGHT$ Statement (standard version only)

Assigns part of a string variable to another string.

RIGHT$(*variable*[**,***length*]**)=***expression*

Parameters

variable
String variable to assign string to.

length
Length of substring.

expression
String to assign.

Remarks

The **RIGHT$** statement replaces the right *length* characters in *variable* with characters from *expression*.

If *length* is omitted the default is 1.

Examples

```
RIGHT$(A$,4)="TEST"
RIGHT$(B$)="0"
```

Mode

Immediate, Program

See Also

MID$**, **RIGHT$

7.198 RSET Statement

Right justifies data in the field variable and moves the data into the file buffer or in a string variable.

RSET *variable***=***expression*

Parameters

variable
Variable to assign value to.

expression
Value to assign to variable.

Examples

```
RSET B$=S$
RSET N$="TEST"
```

Mode

Immediate, Program

See Also

FIELD, **LSET**

7.199 RUN Statement

Runs the current program or a specified program.

RUN [*line*]

- or -

RUN *filespec*

Parameters

line
Line number or label of line to start execution.

filespec
File name of program to load and run.

Remarks

The **RUN** statement clears program memory and closes all open files before running the program.

If *line* is omitted execution begins with the lowest numbered line.

The default extension for the **RUN** statement if not specified in the *filespec* is "nba".

Examples

```
RUN
RUN 100
RUN "ANALYZE.BAS"
```

Mode

Immediate, Program

See Also

CHAIN

7.200 RUNR Statement

Runs a built-in program.

RUNR *program*

Parameters

program
Name of program to load and run.

Remarks

The **RUNR** statement clears program memory and closes all open files before running the program.

Examples

```
RUNR "WELCOME"
```

Mode

Immediate only

See Also

DIRR, **LOADR**

7.201 SAVE Statement

Saves a program.

SAVE *filespec*[**,A**]

- or -

S. *filespec*[**,A**] (Version 1.5 or later)

Parameters

filespec
File name of program to save.

Remarks

The **SAVE** statement can be used to save the current program. The program can be saved in NBASIC binary or ASCII format. By convention, programs in binary format have an extension of "nba" and ASCII format programs have an extension of "bas".

Use **,A** to save the program in ASCII format.

The default extension for the **SAVE** statement if not specified in the *filespec* is "nba".

Examples

```
SAVE "ANALYZE"
SAVE "PROGRAMS:COUNT.BAS",A
```

Mode

Immediate, Program

See Also

ASAVE, **BSAVE**, **LOAD**

7.202 SAVEC Statement

Saves a program to the Windows clipboard.

SAVEC

Remarks

The **SAVEC** statement can be used to save the current program to the Windows clipboard. The program is saved in ASCII format.

Examples

```
SAVEC
```

Mode

Immediate only

See Also

LOADC

7.203 SCREEN BACKUP Statement (Version 1.4 or later)

Saves the screen.

SCREEN [**BACKUP**] *row*,*column*,*columns*,*rows*[,[*foreground*][,*background*]]

Parameters

row
Starting row.

column
Starting column.

columns
Number of columns to fill.

rows
Number of rows to fill.

foreground
Foreground color of saved screen.

background
Background color of saved screen.

Remarks

The **SCREEN BACKUP** statement saves a specified number of rows and columns beginning at *row*,*column*, sets the foreground and background colors to the colors specified and clears the saved part of the screen. Up to 255 screens can be saved. Use the **SCREEN RESTORE** statement to restore a screen.

If *foreground* is omitted the current foreground text color is used. If *background* is omitted the current background text color is used.

Examples

```
SCREEN BACKUP 20,5,40,15
SCREEN BACKUP 20,5,40,15,7
SCREEN BACKUP 20,5,40,15,7,10
SCREEN BACKUP 20,5,40,15,,10
SCREEN 20,5,40,15
```

Mode

Program only

See Also

SCREEN RESTORE, **SCREEN**

7.204 SCREEN RESTORE Statement (Version 1.4 or later)

Restores the screen.

SCREEN [**RESTORE**] [*screen*]

Parameters

screen
Screen to restore.

Remarks

The **SCREEN RESTORE** statement restores a screen saved by the **SCREEN BACKUP** statement.

To restore the last screen saved, use -1. To restore the last two screens, use -2. To restore all screens, omit *screen* or specify 0. To restore a specific screen, use the id of the screen returned by the **SCREEN** function (all screens backed up after this screen will also be restored).

If *screen* is omitted all screens are restored.

Examples

```
SCREEN RESTORE -1
SCREEN RESTORE
SCREEN –2
SCREEN RESTORE 4
```

Mode

Program only

See Also

SCREEN BACKUP, **SCREEN**

7.205 SECURE Statement

Turns secure mode on or off.

SECURE {**ON** | **OFF**}

Remarks

When secure mode is on, statements in a program that may erase or delete files display a prompt allowing the user to abort, retry or ignore the statement. If the statement is aborted, a break at the line containing the statement occurs. If the statement is ignored, the statement is not executed and the program continues with the next statement. Select retry to execute the statement.

The **SECURE** statement is useful to find potentially destructive statements before they are executed, safely allowing further investigation of unfamiliar programs.

Examples

```
SECURE ON
SECURE OFF
```

Mode

Immediate only

See Also

CONFIRM

7.206 SELECT Statement (Version 1.3 or later)

Changes the character set.

SELECT *character-set*

Parameters

character-set
Character set to change to.

Remarks

The **SELECT** statement changes the current character set used to display text on the screen.

To select a character set, specify one of the following values in the **SELECT** statement:

0	ANSI
161	Greek
162	Turkish
177	Hebrew
178	Arabic
186	Baltic
204	Russian
238	East Europe
255	OEM

To make sure the character set was successfully changed, compare the value returned by **PEEK(29538)** to the character set specified in the **SELECT** statement.

On startup, the character set is set to ANSI (0).

Examples

```
SELECT 255
```

Mode

Immediate, Program

See Also

SELECT

7.207 SOUND Statement

Generates a sound through the computer's speaker.

SOUND *frequency*,*duration*

Parameters

frequency

Frequency in hertz of sound.

duration
Number of milliseconds to generate sound.

Remarks

The frequency range is 37Hz to 32,767Hz.

The **SOUND** statement works only on Windows NT/2000 or greater host operating systems. On Windows 95/98 the default sound event is played (similar to **BEEP**).

Examples

```
SOUND 750,100
```

Mode

Immediate, Program

See Also

BEEP

7.208 SPLITNAME Statement (standard version only)

Splits a file specification.

SPLITNAME *filespec*,[*volume*][,[*name*][,*extension*]]

Parameters

filespec
Name of file.

volume
Variable in which the volume is stored.

name
Variable in which the name is stored.

extension
Variable in which the extension is stored.

Remarks

The **SPLITNAME** statement splits a file specification into the volume, filename and extension and stores each component into the respective variable if specified.

If the *volume*, *name* or *extension* variables are omitted that component is not stored.

Examples

```
SPLITNAME "EXAMPLES:DATA.NBA",V$,N$,E$
```

Mode

Immediate, Program

See Also

MAKENAME$**, **SPLITNAME$

7.209 STEP Statement, debugging (standard version only)

Advances execution of the current program to the next statement or line or to a specified line.

STEP

- or -

STEP NEXT

- or -

STEP *line*

Parameters

line
Line number of line where program execution will stop.

Remarks

The **STEP** statement can be used to step through the execution of a program being tested. After the program advances, the program stops, and control is returned to immediate mode.

The first form executes the program and stops at the next statement. The second form executes the program and stops at the next line. The third form executes the program and stops at a specified line number.

The **CONT** statement can be used to continue execution after a breakpoint stops a program.

Debugging must be turned on to step program execution.

Examples

```
STEP
STEP NEXT
STEP 200
```

Mode

Immediate only

See Also

DEBUG

7.210 STOP Statement

Stops a program.

STOP

Remarks

The **STOP** statement stops the currently executing program or immediate mode command.

The **CONT** statement can be used to continue execution after a program stops.

Mode

Immediate, Program

See Also

END

7.211 SWAP Statement (standard version only)

Swaps the contents of a variable with another variable of the same type.

SWAP *variable1*,*variable2*

Parameters

variable1
Any variable.

variable2
Any variable with same type as *variable1*.

Examples

```
SWAP A,B
SWAP T,ITEMS(1)
SWAP N$,USER$
```

Mode

Immediate, Program

7.212 SWAP Statement, editing (standard version only)

Swaps a line in the current program with another line.

SWAP *line1*,*line2*

Parameters

line1
Line number of first line to swap.

line2
Line number of second line to swap.

Examples

```
SWAP 10,20
```

Mode

Immediate only

See Also

APPEND, **COPY**, **MOVE**

7.213 SYSTEM Statement (Version 1.5 or later)

Ends a program.

SYSTEM

Remarks

The **SYSTEM** statement ends the currently executing program or immediate mode command line, closes all open files and closes the printer.

The **SYSTEM** statement is equivalent to the **END** statement and is provided for compatibility.

Mode

Immediate, Program

See Also

STOP

7.214 TEXT Statement (Version 1.4 or later, standard version only)

Turns graphics off.

TEXT

Remarks

The **TEXT** statement is equivalent to the **GRAPH OFF** statement and is provided for compatibility.

Mode

Program only

See Also

GR, **GRAPH**, **HGR**, **HGR2**

7.215 TIMER Statement

Enables, disables or suspends timer event trapping or sets the value of the timer.

TIMER {**ON** | **OFF** | **STOP**}

- or -

TIMER=*timer*

Parameters

timer
New value of timer in milliseconds.

Remarks

The first form enables, disables, or suspends timer event trapping. The second form sets the value of the timer. The timer begins at 0 and represents the elapsed time in milliseconds since NBASIC was started.

Examples

```
TIMER ON
TIMER OFF
TIMER STOP
TIMER=0
```

Mode

Immediate, Program

See Also

ON TIMER

7.216 TRACE Statement

Turns tracing of program on or off.

TRACE {**ON** | **OFF**}

- or -

TRACE (Version 1.4 or later)

Examples

```
TRACE ON
TRACE OFF
TRACE
```

Mode

Immediate, Program

See Also

TROFF, **TRON**

7.217 TROFF Statement

Disables tracing of program.

TROFF

Mode

Immediate, Program

See Also

TRACE, **TRON**

7.218 TRON Statement

Enables tracing of program.

TRON

Mode

Immediate, Program

See Also

TRACE, **TROFF**

7.219 TRUNCATE Statement

Truncates one or more open files to zero length.

TRUNCATE

- or -

TRUNCATE #*file*[**,**#*file*]...

Parameters

file
File number of open file to truncate.

Remarks

The **TRUNCATE** statement sets the length of a file to 0, discarding the file's contents. Files opened for INPUT cannot be truncated.

TRUNCATE with no parameters truncates all open files.

Examples

```
TRUNCATE
TRUNCATE #1
```

Mode

Immediate, Program

See Also

OPEN

7.220 TYPE Statement (standard version only)

Displays the contents of a file.

TYPE *filespec*

Parameters

filespec
Name of file to type.

Remarks

The **TYPE** statement displays the file contents as ASCII.

Examples

```
TYPE "LIST.TXT"
```

Mode

Immediate only

See Also

DUMP

7.221 UNBREAK Statement (Version 1.4 or later, standard version only)

Clears breakpoints.

UNBREAK

- or -

UNBREAK *line*[**,***line*]...

- or -

UNBREAK [*line*][**-**[*line*]][,[*line*][**-**[*line*]]]...

Parameters

line
Line number or label where breakpoint is to be cleared.

Remarks

The **UNBREAK** statement can be used to clear breakpoints set by **BREAK**.

UNBREAK with no parameters clears all breakpoints.

The **UNBREAK** statement is equivalent to the **NOBREAK** statement and is provided for compatibility.

Examples

```
UNBREAK
UNBREAK 100,150
UNBREAK 200,*REDO
UNBREAK -10,20,50-100,900-
```

Mode

Immediate, Program

See Also

BREAK, **NOBREAK**

7.222 UNLOAD Statement

Closes all open files on a specified volume.

UNLOAD *volume*

- or -

UNLOAD [*volume*] (Version 1.4 or later)

Parameters

volume
Volume on which all open files will be closed.

Remarks

The **UNLOAD** statement writes any remaining data in the file buffer to the file and closes the file. All open files on the specified volume are closed.

If *volume* is omitted, all open files on the default volume are closed.

Examples

```
UNLOAD "HOME"
UNLOAD
```

Mode

Immediate, Program

See Also

CLOSE, **OPEN**

7.223 UNLOCK Statement

Unlocks the screen.

UNLOCK #0

Remarks

The **UNLOCK** statement unlocks the screen and displays changes to the screen made after the **LOCK** statement.

Mode

Program only

See Also

LOCK

7.224 UNNUM Statement (Version 1.5 or later)

Unnumbers lines in the current program.

UNNUM

Remarks

The **UNNUM** statement removes line numbers of lines that are not referenced in the current program.

Mode

Immediate only

See Also

RENUM

7.225 UNREMARK Statement (standard version only)

Unremarks (uncomments) lines in the current program.

UNREMARK {*line*[**-***line*] | *-line*}

- or -

UNREMARK {*line*[**-***line*] | *-line*}[,{*line*[**-***line*] | *-line*}]... (Version 1.2 or later)

Parameters

line
Line number of line to unremark.

Remarks

The **UNREMARK** statement removes the **REM** statement from the beginning of the specified lines, which allows the lines to be executed.

Examples

```
UNREMARK 100
UNREMARK 10-90
UNREMARK -200
UNREMARK -10,20,50-100,900-
```

Mode

Immediate only

See Also

REMARK

7.226 UNTRACE Statement (Version 1.4 or later)

Disables tracing of program.

UNTRACE

Mode

Immediate, Program

Remarks

The **UNTRACE** statement is equivalent to the **TROFF** statement and is provided for compatibility.

See Also

TRACE, **TROFF**

7.227 VER Statement

Displays version number.

VER

Mode

Immediate, Program

See Also

ABOUT

7.228 VLIN Statement (Version 1.4 or later, standard version only)

Draws a vertical line.

VLIN *y1*,*y2* **AT** *x*

Parameters

y1
Y coordinate of start of line.

y1
Y coordinate of end of line.

x
X coordinate of line.

Remarks

The **VLIN** statement draws a line from the point (x,y1) to the point (x,y2) using the current graphics foreground color.

The **VLIN** statement is similar to the **LINE** statement and is provided for compatibility.

Graphics must be turned on to draw lines.

Examples

```
VLIN 10,20 AT 10
```

Mode

Program only

See Also

COLOR, **GR**, **HLIN**, **LINE**

7.229 VOLINI Statement

Deletes all files on a volume.

VOLINI *volume*

Parameters

volume
Volume to initialize.

Remarks

NOTE: This permanently deletes all files from the volume.

Examples

```
VOLINI "TEMP"
```

Mode

Immediate, Program

7.230 VOLUME Statement

Makes a specified volume the default volume.

VOLUME *volume*

- or -

V. *volume* (Version 1.5 or later)

Parameters

volume
Volume to make the default volume.

Remarks

NOTE: If the volume is omitted from a file name the default volume is used.

Examples

```
VOLUME "EXAMPLES"
```

Mode

Immediate, Program

See Also

DIR, **VOLUMES**

7.231 VOLUMES Statement

Displays the current mapped volumes.

VOLUMES

Mode

Immediate only

See Also

DIR, **VOLUME**

7.232 VSCROLL Statement (standard version only)

Scrolls rows up or down.

VSCROLL *row*,*column*,*columns*,*rows*[,*vertical*]

Parameters

row
Starting row.

column
Starting column.

columns
Number of columns to scroll.

rows
Number of rows to scroll.

vertical
Number of rows by which to scroll.

Remarks

The **VSCROLL** statement scrolls text within a rectangular region *length* columns by *height* rows starting at *row*,*column*, *vertical* rows. If *vertical* is positive the region is scrolled up, if negative down. The text is scrolled within the rectangle with rows above or below overwritten and new rows filled with spaces.

If *vertical* is omitted the default is 1.

Examples

```
VSCROLL 0,0,80,25
VSCROLL 0,0,80,25,-1
```

Mode

Immediate, Program

See Also

HSCROLL

7.233 VTAB Statement (Version 1.4 or later)

Moves the cursor to a specified position on the screen.

VTAB *row*

Parameters

row
Row where cursor is to be moved.

Remarks

The **VTAB** statement only changes the row position of the cursor.

The screen has 25 rows numbered 0 through 24.

The **VTAB** statement is similar to the **LOCATE** statement and is provided for compatibility.

Examples

```
VTAB 10
```

Mode

Immediate, Program

See Also

HTAB, **LOCATE**

7.234 WAIT Statement

Pauses a program.

WAIT [*duration*]

Parameters

duration
Number of milliseconds to pause.

Remarks

If *duration* is omitted the default is 1000 milliseconds.

Examples

```
WAIT
WAIT 2000
```

Mode

Immediate, Program

See Also

PAUSE

7.235 WRITE Statement

Writes output to the screen, a file or the printer.

WRITE [#*file*,] *expression*[,*expression*]...

Parameters

file

File number of open file or printer (-2).

expression

Data to print.

Remarks

The **WRITE** statement inserts commas between items and quotation marks around strings, which can be read by the **INPUT** statement.

Use **WRITE #-2** to write output to the printer.

Examples

```
WRITE "ABC",123
WRITE #1, 1000,"TEST"
```

Mode

Immediate, Program

See Also

OPEN, **PRINT**

8
Functions

8.1 ABS Function

Returns the absolute value of a number.

ABS(*number***)**

Arguments

number
Number whose absolute value will be returned.

Examples

```
ABS(10)          returns 10
ABS(-10)         returns 10
```

See Also

SGN

8.2 ACCESS Function

Determines if a file is accessible in a given mode.

ACCESS(*filename***,***mode***)**

Arguments

filename
Name of file to check.

mode
Mode of file to check access for.

Remarks

The **ACCESS** function determines if a file can be opened with the specified mode. The mode argument can be one of the following:

-1	checks for existence only
0	checks if the file can be opened for append
1	checks if the file can be opened for input
2	checks if the file can be opened for output
3	checks if the file can be opened for random

Examples

```
ACCESS("HOME:ACCOUNT.DAT",3)
```
returns -1 (true) if the file HOME:ACCOUNT.DAT can be opened in random mode, otherwise returns 0 (false).

See Also

OPEN, **EXISTS**

8.3 ACOS Function

Returns the arccosine of a number.

ACOS(*number***)**

Arguments

number
Number whose arccosine will be returned.

Remarks

The **ACOS** function returns the angle in radians whose cosine is *number* (-1 <= *number* <= 1).

Examples

```
ACOS(0)    returns 1.5707963267949
ACOS(1)    returns 0
```

See Also

COS

8.4 ADJUST Function

Adjusts a time value.

ADJUST(*time***,***seconds***)**

Arguments

time
Time value to be adjusted.

seconds
Number of seconds by which to adjust time.

Remarks

The **ADJUST** function can be used to adjust a time value a specified number of seconds. The value of *seconds* can be positive to adjust forward or negative to adjust backward.

Examples

```
ADJUST(NOW,10)          returns the current time + 10 seconds
ADJUST(NOW,-10)         returns the current time - 10 seconds
```

See Also

NOW

8.5 AFTER$ Function (standard version only)

Returns part of a string.

AFTER$(*string*,*index***)**

Arguments

string
String from which part will be returned.

index
Index after which part to be returned starts.

Examples

```
AFTER$("ABC123",3)          returns "123"
```

See Also

BEFORE$

8.6 ALNUM Function

Determines if a character is alphanumeric.

ALNUM(*character***)**

Arguments

character
Character to check.

Remarks

The **ALNUM** function checks if a character is alphanumeric (A-Z, a-z, 0-9).

Examples

```
ALNUM("A")      returns -1 (true)
ALNUM("1")      returns -1 (true)
ALNUM("$")      returns 0 (false)
```

See Also

ALPHA, **CNTRL**, **DIGIT**, **HEX**, **LOWER**, **PRINT**, **UPPER**

8.7 ALPHA Function

Determines if a character is alphabetic.

ALPHA(*character***)**

Arguments

character
Character to check.

Remarks

The **ALPHA** function checks if a character is alphabetic (A-Z, a-z).

Examples

```
ALPHA("A")      returns -1 (true)
ALPHA("1")      returns 0 (false)
ALPHA("$")      returns 0 (false)
```

See Also

ALNUM, **CNTRL**, **DIGIT**, **HEX**, **LOWER**, **PRINT**, **UPPER**

8.8 ASC Function

Returns the ordinal value of a character.

ASC(*character***)**

Arguments

character
Character whose ordinal value will be returned.

Remarks

The **ASC** function is equivalent to the **ORD** function and is provided for compatibility.

Examples

```
ASC("A")        returns 65
ASC("1")        returns 49
ASC("$")        returns 36
```

See Also

ORD

8.9 ASIN Function

Returns the arcsine of a number.

ASIN(*number***)**

Arguments

number
Number whose arcsine will be returned.

Remarks

The **ASIN** function returns the angle in radians whose sine is *number* (-1 <= *number* <= 1).

Examples

```
ASIN(0)   returns 0
```

```
ASIN(1)    returns 1.5707963267949
```

See Also

SIN

8.10 ATN Function

Returns the arctangent of a number.

ATN(*number***)**

Arguments

number
Number whose arctangent will be returned.

Remarks

The **ATN** function returns the angle in radians whose tangent is *number*.

Examples

```
ATN(0)     returns 0
ATN(1)     returns 0.463647609000806
```

See Also

TAN

8.11 ATTRIB$ Function

Returns the read-only attribute of a file.

ATTRIB$(*filename***)**

Arguments

filename
Name of file to whose read-only attribute will be returned.

Examples

```
ATTRIB$("EXAMPLES:DATA.NBA")
```
returns "R" if the file EXAMPLES:DATA.NBA is read-only, otherwise returns "".

See Also

ATTRIB

8.12 BEFORE$ Function (standard version only)

Returns part of a string.

BEFORE$(*string***,***index***)**

Arguments

string
String from which part will be returned.

index
Index before which part to be returned ends.

Examples

```
BEFORE$("ABC123",3)          returns "AB"
```

See Also

AFTER$

8.13 BEGINSWITH Function (Version 1.2 or later, standard version only)

Determines if a string begins with another string.

BEGINSWITH(*string1***,***string2***)**

Arguments

string1
String to check.

string2
String to compare to beginning.

Remarks

The **BEGINSWITH** function is case sensitive.

Examples

```
BEGINSWITH("ABCDEF","ABC")        returns -1 (true)
BEGINSWITH("ABCDEF","abc")        returns 0 (false)
```

See Also

ENDSWITH

8.14 BIN$ Function

Returns a string representing the binary value of a number.

BIN$(*number***)**

Arguments

number
Number whose binary string representation will be returned.

Examples

```
BIN$(35)          returns "100011"
```

See Also

HEX$**, **OCT$, **STR$**

8.15 BREAK Function (standard version only)

Returns the index of the first occurrence of any character from a string in another string.

BREAK(*string1***,***string2***)**

Arguments

string1
String to search.

string2
Characters to search for.

Remarks

The **BREAK** function searches *string1* for characters in *string2* and returns the index of the first character found.

Examples

BREAK("ABCDEF","CD") returns 3

8.16 BUFSIZ Function (Version 1.5 or later)

Returns the buffer length of an open file.

BUFSIZ(*file***)**

Arguments

file
Number of open file whose buffer length will be returned.

Remarks

The buffer length of a file is specified in the **OPEN** statement.

Examples

`BUFSIZ(1)` returns the buffer length

See Also

OPEN

8.17 CBR Function

Returns the cube root of a number.

CBR(*number***)**

Arguments

number

Number whose cube root will be returned.

Examples

```
CBR(27)          returns 3
CBR(-27)         returns -3
```

See Also

CUBE, **SQR**

8.18 CBRT Function

Returns the cube root of a number.

CBRT(*number***)**

Arguments

number
Number whose cube root will be returned.

Remarks

The **CBRT** function is equivalent to the **CBR** function and is provided for compatibility.

Examples

```
CBRT(27)         returns 3
CBRT(-27)        returns -3
```

See Also

CBR

8.19 CDN Function

Converts a decimal digit to a number.

CDN(*character***)**

Arguments

character
Decimal digit to convert.

Remarks

The **CDN** function converts a string containing a decimal digit (0-9) to a number.

Examples

```
CDN("1")         returns 1
```

See Also

CHN

8.20 CEIL Function

Returns the ceiling of a number.

CEIL(*number***)**

Arguments

number
Number whose ceiling will be returned.

Remarks

The **CEIL** function returns the smallest integer that is greater than or equal to *number*.

Examples

```
CEIL(1.8)       returns 2
CEIL(-1.8)      returns -1
```

See Also

INT

8.21 CENTER$ Function (standard version only)

Returns a string centered in a specified number of spaces.

CENTER$(*string***,***length***[,***pad***])**

Arguments

string
String which to center.

length
Number of spaces in which string will be centered.

pad
Character to pad centered string with.

Remarks

If *pad* is omitted the default is a space.

Examples

```
CENTER$("TEST",10)          returns "  TEST  "
CENTER$("TEST",10,"*")      returns "***TEST***"
```

8.22 CHANGE$ Function (standard version only)

Changes each occurrence of a string within another string to another string.

CHANGE$(*string1***,***string2***,***string3***)**

Arguments

string1
String in which to search.

string2
String to change.

string3
New string.

Examples

```
CHANGE$("ABC123","ABC","XYZ")          returns "XYZ123"
```

See Also

REMOVE$

8.23 CHARS$ Function (standard version only)

Returns a string containing every character (&H00-&HFF).

CHARS$

8.24 CHN Function

Converts a hexadecimal digit to a number.

CHN(*character***)**

Arguments

character
Hexadecimal digit to convert.

Remarks

The **CHN** function converts a string containing a hexadecimal digit (0-9,A-F) to a number.

Examples

```
CHN("1")      returns 1
CHN("A")      returns 10
CHN("f")      returns 15
```

See Also

CDN

8.25 CHOOSE Function (standard version only)

Selects a number.

CHOOSE(*index***,***number***[,***number***]...)**

Arguments

index
Index of number to select.

number
List of numbers to select from.

Remarks

The **CHOOSE** function selects a *number* at a specified *index* in a list. The list index begins at 1 and the list can contain one or more numbers.

Examples

```
CHOOSE(2,10,20,30)          returns 20
```

See Also

CHOOSE$, **SWITCH**, **SWITCH$**

8.26 CHOOSE$ Function (standard version only)

Selects a string.

CHOOSE$(*index***,***string*[**,***string*]...**)**

Arguments

index
Index of string to select.

stringr
List of strings to select from.

Remarks

The **CHOOSE$** function selects a *string* at a specified *index* in a list. The list index begins at 1 and the list can contain one or more strings.

Examples

```
CHOOSE$(2,"A","B","C")          returns "B"
```

See Also

CHOOSE$, **SWITCH**, **SWITCH$**

8.27 CHR$ Function

Returns a string containing the character associated with a character code.

CHR$(*number***)**

- or -

CHR$(*number*[**,***number*]...**)** (Version 1.4 or later)

Arguments

number
Number whose character association will be returned.

Examples

```
CHR$(65)        returns "A"
```

See Also

ORD

8.28 CHRS$ Function (Version 1.4 or later)

Returns a string containing the character associated with a character code.

CHRS$(*number***)**

Arguments

number
Number whose character association will be returned.

Remarks

The **CHRS$** function is equivalent to the **CHR$** function and is provided for compatibility.

Examples

```
CHRS$(65)           returns "A"
```

See Also

ORD

8.29 CLEAN$ Function (standard version only)

Removes all non-printable characters from a string.

CLEAN$(*string***)**

Arguments

string
String to be cleaned.

Remarks

The **CLEAN$** function removes control characters (&H00-&H1F,&H7F) from a string.

8.30 CNTRL Function

Determines if a character is a control character.

CNTRL(*character***)**

Arguments

character
Character to check.

Remarks

The **CNTRL** function checks if a character is a control character (&H00-&H1F,&H7F).

Examples

```
CNTRL("A")          returns 0 (false)
CNTRL("1")          returns 0 (false)
CNTRL(CHR$(&H10))   returns -1 (true)
```

See Also

ALNUM, **ALPHA**, **DIGIT**, **HEX**, **LOWER**, **PRINT**, **UPPER**

8.31 COLOR Function

Returns the current text color.

COLOR[**(***color***)**]

Arguments

color
Specifies which text color to return.

Remarks

If *color* is greater than or equal to 0, the current foreground text color is returned, otherwise the current background text color is returned.

If *color* is omitted, the current foreground text color is returned.

Examples

```
COLOR          returns the current foreground text color
COLOR(-1)      returns the current background text color
```

See Also

COLOR

8.32 COLUMN Function

Returns the column position of the cursor.

COLUMN

See Also

ROW

8.33 COMB Function

Returns the number of combinations.

COMB(*n*,*k***)**

Arguments

n
Number of objects.

k
Number of objects in each set.

Remarks

The **COMB** function calculates the number of combinations of *n* objects taken *k* at a time (*n*!/*k*!(*n*-*k*)!).

Examples

`COMB(5,2)` returns 10

See Also

PERM

8.34 COMP Function (Version 1.4 or later)

Compares numbers or strings.

COMP(*value1*,*value2***)**

Arguments

value1
First value to compare.

value2
Second value to compare.

Remarks

The **COMP** function returns 0 if the values are identical, 1 if *value1* is greater than *value2* or -1 if *value1* is less than *value2*.

String comparisons are case sensitive.

Examples

`COMP("abc","def")`	returns `-1`
`COMP("A","A")`	returns 0
`COMP(2,1)`	returns 1

See Also

COMPI

8.35 COMPI Function (Version 1.5.2 or later)

Compares numbers or strings.

COMPI(*value1***,***value2***)**

Arguments

value1
First value to compare.

value2
Second value to compare.

Remarks

The **COMPI** function returns 0 if the values are identical, 1 if *value1* is greater than *value2* or -1 if *value1* is less than *value2*.

Numeric comparisons are absolute and string comparisons are case insensitive.

Examples

```
COMPI("abc","def")    returns -1
COMPI("A","a")        returns 0
COMPI(2,-2)           returns 1
```

See Also

COMP

8.36 COMPRESS$ Function

Trims spaces from the left and right of a string and compresses duplicate spaces to a single space.

COMPRESS$(*string***)**

Arguments

string
String to be compressed.

Examples

```
COMPRESS$(" A     B C ")      returns "A B C"
```

See Also

LTRIM$**, **RTRIM$, **TRIM$**

8.37 COPYRIGHT$ Function

Returns the copyright information.

COPYRIGHT$

8.38 COPYSIGN Function

Returns one value with the sign of another.

COPYSIGN(*number1***,***number2***)**

Arguments

number1
Value to return.

number2
Value of sign.

Examples

```
COPYSIGN(10,-1)      returns -10
COPYSIGN(-5,1)       returns 5
COPYSIGN(2,1)        returns 2
```

8.39 COS Function

Returns the cosine of an angle.

COS(*angle***)**

Arguments

angle
Angle in radians whose cosine will be returned.

Examples

```
COS(0)          returns 1
COS(PI/4)       returns 0.707106781186548
```

See Also

ACOS, **COSH**, **SIN**

8.40 COSH Function

Returns the hyperbolic cosine of an angle.

COSH(*angle***)**

Arguments

angle
Angle in radians whose hyperbolic cosine will be returned.

Examples

```
COSH(PI)        returns 11.5919532755215
```

See Also

COS, **SINH**

8.41 COT Function

Returns the cotangent of an angle.

COT(*angle***)**

Arguments

angle
Angle in radians whose cotangent will be returned.

Remarks

The **COT** function returns the cotangent (1/tan(*angle*)) of *angle*.

Examples

```
COT(PI/4)
```
returns 1

See Also

TAN

8.42 COUNT Function

Counts the occurrences of a string within another string.

COUNT(*string1***,***string2***)**

Arguments

string1
String in which to search.

string2
String whose number of occurrences in the first string will be returned.

Examples

COUNT("A STRING","RING") returns 1
COUNT("A STRING","CUP") returns 0

8.43 CSC Function

Returns the cosecant of an angle.

CSC(*angle***)**

Arguments

angle
Angle in radians whose cosecant will be returned.

Remarks

The **CSC** function returns the cosecant (1/sin(*angle*)) of *angle*.

Examples

```
CSC(PI/2)          returns 1
```

See Also

SEC

8.44 CSPAN Function (standard version only)

Returns the length of a substring beginning a string not containing characters in another string.

CSPAN(*string1***,***string2***)**

Arguments

string1
String to search.

string2
Characters to search for.

Examples

CSPAN("ABABCDCD","CD") returns 4

See Also

SPAN

8.45 CSRLIN Function

Returns the row position of the cursor.

CSRLIN

Remarks

The **CSRLIN** function is equivalent to the **ROW** function and is provided for compatibility.

See Also

ROW

8.46 CTIME$ Function

Returns a string containing the date and time of a time value.

CTIME$(*time***)**

Arguments

time
Time value whose date and time will be returned.

Examples

```
CTIME$(12672412248)          returns "Mon Jul 29 10:30:48 2002"
```

See Also

NOW

8.47 CTRL$ Function (Version 1.2 or later)

Returns the CTRL key.

CTRL$

Remarks

The **CTRL$** function returns a string containing the character codes for the CTRL key (&H00, &H11).

8.48 CUBE Function

Returns the cube of a number.

CUBE(*number***)**

Arguments

number
Number whose cube will be returned.

Examples

```
CUBE(3)          returns 27
CUBE(-3)         returns -27
```

See Also

CBR

8.49 CVN Function

Converts a binary string to a number.

CVN(*string***)**

Arguments

string
Binary string to convert.

Remarks

The **CVN** function converts a binary string created with the **MKN$** function to a number.

See Also

MKN$

8.50 DATE Function

Returns the current date as a number in the form YYYYDDD.

DATE

See Also

DATE$

8.51 DATE$ Function

Returns the date as a string in the form MM-DD-YYYY.

DATE$[**(***time***)**]

Arguments

time
Time value whose date will be returned.

Remarks

If *time* is omitted the current date is returned.

Examples

`DATE$(MKTIME(2002,7,24))` returns "07-24-2002"

See Also

DATE, **TIME$**

8.52 DAY Function

Returns the day of the month from a time value.

DAY(*time***)**

Arguments

time
Time value whose day of the month will be returned.

Examples

`DAY(MKTIME(2002,7,24))` returns 24

See Also

HOUR, **MINUTE**, **MONTH**, **SECOND**, **WEEKDAY**, **YEAR**

8.53 DAYNAME$ Function

Returns the day name from a time value.

DAYNAME$(*day*[**,***format*]**)**

Arguments

day
Day of week whose name will be returned.

format
Format of day name.

Remarks

If *format* is omitted the long name of the day is returned. If *format* is non-zero the short name of the day is returned.

Examples

```
DAYNAME$(1)          returns "Monday"
DAYNAME$(1,1)        returns "Mon"
```

See Also

MONTHNAME$

8.54 DEBUG Function (standard version only)

Determines if debug mode is on or off.

DEBUG

Remarks

The **DEBUG** function returns -1 (true) if debug mode is on, 0 (false) otherwise.

See Also

DEBUG

8.55 DEC Function

Converts a string to a number.

DEC(*string*[**,***base*]**)**

Arguments

string
String to convert.

base
Base of *number*.

Remarks

The **DEC** function can convert strings from base 2 (binary), base 8 (octal), base 10 (decimal), and base 16 (hexadecimal).

If *base* is omitted the default is 10 (decimal).

Examples

```
DEC("100")          returns 100
DEC("100",2)        returns 4
DEC("100",8)        returns 64
DEC("100",10)       returns 100
DEC("100",16)       returns 256
```

See Also

BIN$**, **HEX$, **OCT$**, **STR$**, **VAL**

8.56 DEFAULT Function

Determines if a volume is the default.

DEFAULT(*volume***)**

Arguments

volume
Volume to check.

Examples

`DEFAULT("HOME")` returns -1 (true) if HOME is the default volume, otherwise returns 0 (false).

See Also

MAPPED, **READONLY**

8.57 DEG Function

Converts radians to degrees.

DEG(*radians***)**

Arguments

radians
Radians whose value in degrees will be returned.

Examples

`DEG(PI)` returns 180

See Also

RAD

8.58 DELETE$ Function (standard version only)

Deletes part of a string.

DELETE$(*string***,***index***,***length***)**

Arguments

string
String from which part will be deleted.

index
Start of substring to delete.

length
Number of characters in substring to delete.

Examples

```
DELETE$("ABC123",3,2)          returns "AB23"
```

See Also

INSERT$**, **REPLACE$

8.59 DIGIT Function

Determines if a character is numeric digit.

DIGIT(*character***)**

Arguments

character
Character to check.

Remarks

The **DIGIT** function checks if a character is a numeric digit (0-9).

Examples

```
DIGIT("A")      returns 0 (false)
DIGIT("1")      returns -1 (true)
DIGIT("$")      returns 0 (false)
```

See Also

ALNUM, **ALPHA**, **CNTRL**, **HEX**, **LOWER**, **PRINT**, **UPPER**

8.60 DOLLAR$ Function (standard version only)

Formats a number as currency.

DOLLAR$(*amount*[**,***decimals*]**)**

Arguments

amount
Number to be formatted.

decimals
Number of decimal places.

Remarks

The **DOLLAR$** function rounds the number to the specified number of decimal places.

If *decimals* is omitted the default is 2.

Examples

```
DOLLAR$(123.45)          returns "$123.45"
DOLLAR$(123.45,4)        returns "$123.4500"
DOLLAR$(123.45,0)        returns "$123"
DOLLAR$(123.45,-1)       returns "$120"
```

See Also

USING$

8.61 DTR Function

Converts degrees to radians.

DTR(*degrees***)**

Arguments

degrees
Degrees whose value in radians will be returned.

Remarks

The **DTR** function is equivalent to the **RAD** function and is provided for compatibility.

Examples

```
DTR(180)        returns 3.14159265358979
```

See Also

RAD

8.62 EDIT$ Function (standard version only)

Returns the result of editing a string.

EDIT$(*length***,***string***)**

Arguments

length
Number of characters to read.

string
String to be edited.

Remarks

The **EDIT$** function is similar to the **INPUT$** function but initializes the input with *string*.

See Also

INPUT$

8.63 EMPTY Function (Version 1.4 or later)

Checks for 0 or empty string.

EMPTY(*expression***)**

Arguments

expression
Value to check.

Remarks

The **EMPTY** function returns TRUE (-1) if the expression evaluates to 0 or an empty string or FALSE (0) otherwise.

Examples

```
EMPTY(0)        returns -1 (true)
EMPTY(1)        returns 0 (false)
EMPTY("")       returns -1 (true)
EMPTY("A")      returns 0 (false)
```

8.64 ENCLOSE$ Function (Version 1.4 or later)

Encloses a string.

ENCLOSE$(*string*[**,***characters*]**)**

Arguments

string
String to be enclosed.

characters
Characters to enclose string with (2).

Remarks

The **ENCLOSE$** function encloses a string with the characters specified.

if *characters* is omitted the default is double quotes (""""").

The **ENCLOSE$** function is equivalent to the **QUOTE$** function and is provided for compatibility.

Examples

```
ENCLOSE$("hello")               returns ""hello""
```

```
ENCLOSE$("abcd","[]")        returns "[abcd]"
```

See Also

QUOTE$

8.65 ENDSWITH Function (Version 1.2 or later, standard version only)

Determines if a string ends with another string.

ENDSWITH(*string1***,***string2***)**

Arguments

string1
String to check.

string2
String to compare to end.

Remarks

The **ENDSWITH** function is case sensitive.

Examples

```
ENDSWITH("ABCDEF","DEF")          returns -1 (true)
ENDSWITH("ABCDEF","def")          returns 0 (false)
```

See Also

BEGINSWITH

8.66 EOF Function

Returns the end of file status of an open file.

EOF(*file***)**

Arguments

file
Number of open file whose end of file status will be returned.

Examples

```
EOF(1)
```
returns -1 if end of file, 0 otherwise.

See Also

LOC, **LOF**, **REC**

8.67 ERL Function

Returns the number of the line where the last error occurred.

ERL

See Also

ERR, **ERR$**

8.68 ERLIN Function (Version 1.5 or later)

Returns the number of the line where the last error occurred.

```
ERLIN
```

Remarks

The **ERLIN** function is equivalent to the **ERL** and is provided for compatibility.

See Also

ERL

8.69 ERN Function

Returns the error number of the last error to occur.

```
ERN
```

Remarks

The **ERN** function is equivalent to the **ERR** function and is provided for compatibility.

See Also

ERR

8.70 ERNO Function (Version 1.5 or later)

Returns the error number of the last error to occur.

```
ERNO
```

Remarks

The **ERNO** function is equivalent to the **ERR** function and is provided for compatibility.

See Also

ERR

8.71 ERR Function

Returns the error number of the last error to occur.

```
ERR
```

See Also

ERL, **ERR$**

8.72 ERR$ Function

Returns the description of an error code.

ERR$(*error***)**

Arguments

error
Error code whose description will be returned.

Examples

```
ERR$(16)          returns "File not found"
```

See Also

ERL, **ERR**

8.73 ESC$ Function

Returns the ESCAPE character (decimal 27).

ESC$

8.74 EVAL Function (standard version only)

Evaluates an expression and returns the result.

EVAL(*expression***)**

Arguments

expression
Expression to evaluate.

Remarks

The **EVAL** function evaluates a string containing a valid expression and returns the result. The *expression* must evaluate to a numeric value and can contain numbers, operators, built-in functions, user-defined functions, and variables.

Examples

```
EVAL("1+2")                 returns 3
EVAL("SQRT(9)*2")           returns 6
EVAL("LEN(""TEST"")")       returns 4
```

8.75 EVEN Function (standard version only)

Determines if a number is even.

EVEN(*number***)**

Arguments

number
Number to check.

Examples

```
EVEN(2)          returns -1 (true)
EVEN(3)          returns 0 (false)
```

See Also

ODD

8.76 EXISTS Function

Determines if a file exists.

EXISTS(*filename***)**

Arguments

filename
Name of file to check.

Examples

```
EXISTS("EXAMPLES:DATA.NBA")      returns -1 (true) if the file
                                 EXAMPLES:DATA.NBA exists,
                                 otherwise returns 0 (false).
```

See Also

ACCESS

8.77 EXP Function

Returns the natural exponential of a number.

EXP(*number***)**

Arguments

number
Number whose natural exponential will be returned.

Remarks

The **EXP** function returns the natural exponential (e^{number}) of *number*.

Examples

```
EXP(2)           returns 7.38905609893068
```

See Also

EXP2, **EXP10**

8.78 EXP2 Function

Returns the base 2 exponential of a number.

EXP2(*number***)**

Arguments

number
Number whose base 2 exponential will be returned.

Remarks

The **EXP2** function returns the base 2 exponential (2^{number}) of *number*.

Examples

`EXP2(2)` returns 4

See Also

EXP, **EXP10**

8.79 EXP10 Function

Returns the base 10 exponential of a number.

EXP10(*number***)**

Arguments

number
Number whose base 10 exponential will be returned.

Remarks

The **EXP10** function returns the base 10 exponential (10^{number}) of *number*.

Examples

`EXP10(2)` returns 100

See Also

EXP, **EXP2**

8.80 EXTRACT$ Function (standard version only)

Returns part of a string before the first occurrence of another string.

EXTRACT$(*string1***,***string2***)**

Arguments

string1
String from which a part will be returned.

string2

String after which the part to be returned ends.

Examples

EXTRACT$("ABCDEF","CD") returns "AB"

See Also

REMAIN$

8.81 FACT Function (standard version only)

Calculates the factorial of a number.

FACT(*number***)**

Arguments

number
Number whose factorial will be returned.

Remarks

The **FACT** function returns the factorial (*number* * (*number* - 1) * ... * 1) of *number*.

Examples

```
FACT(3)          returns 6
FACT(4)          returns 24
```

8.82 FALSE Function

Returns false (0).

FALSE

See Also

TRUE

8.83 FILE$ Function

Enumerates files on a volume.

FILE$(*filespec***,***control***)**

Arguments

filespec
Files to enumerate.

control
Determines what file will be returned.

Remarks

If *control* is 0 file enumeration is initialized and the name of the first file is returned. Specify an empty string for *filespec* and 1 for *control* to return the next file name. An

empty string is returned when there are no more files. The files are returned in no particular order.

Examples

`FILE$("EXAMPLES:*.BAS",0)`	returns the first file name matching EXAMPLES:*.BAS or an empty string if no matching files found.
`FILE$("",1)`	returns the next matching file or an empty string if no more matching files found.

8.84 FILEINFO$ Function (standard version only)

Returns file information.

FILEINFO$(*filespec***)**

Arguments

filespec
Name of file to return information for.

Remarks

The **FILEINFO$** function returns the file information for the specified file. The return string is the same format used by the default format of the **DIR** statement (except the file size does not contain commas). The information includes the 8.3 name of the file (first 8 characters), a space, the 8.3 extension (3 characters), a space, the file size (14 characters, right justified), a space, the modification date and time (mm/dd/yyyy hh:mmpm), a space and the long file name.

If *filespec* does not exist an empty string is returned.

Examples

`FILEINFO$("EXAMPLES:NONUM.BAS")` returns "NONUM BAS 224 10/15/2003 8:56pm nonum.bas"

See Also

DIR, **FILE$**, **VOLINFO$**

8.85 FILEMODE Function

Returns the mode of an open file.

FILEMODE(*file***)**

Arguments

file
Number of open file whose mode will be returned.

Remarks

The **FILEMODE** function returns 0 for append, 1 for input, 2 for output, or 3 for random.

See Also

FILEMODE$

8.86 FILEMODE$ Function

Returns the mode of an open file.

FILEMODE$(*file***)**

Arguments

file
Number of open file whose mode will be returned.

Remarks

The **FILEMODE$** function returns "A" for append, "I" for input, "O" for output, or "R" for random.

See Also

FILEMODE

8.87 FILENAME$ Function

Returns the file name of an open file.

FILENAME$(*file***)**

Arguments

file
Number of open file whose file name will be returned.

Remarks

The **FILENAME$** function returns the file name of the file including the volume.
+
See Also

OPEN

8.88 FIND Function (standard version only)

Finds the position of a character within a string.

FIND(*string***,***character*[**,***index*]**)**

Arguments

string
String in which to search.

character
Character to search for.

index

Position where search is to begin.

Remarks

If *character* is not found 0 is returned.

If *index* is omitted the default is 1.

Examples

```
FIND("A STRING","S")        returns 3
FIND("A STRING","S",4)      returns 0
```

See Also

RFIND

8.89 FINDONEOF Function (standard version only)

Returns the index of the first occurrence of any character from a string in another string.

FINDONEOF(*string1***,***string2***)**

Arguments

string1
String to search.

string2
Characters to search for.

Remarks

The **FINDONEOF** function searches *string1* for characters in *string2* and returns the index of the first character found.

The **FINDONEOF** function is equivalent to the **BREAK** function and is provided for compatibility.

Examples

FINDONEOF("ABCDEF","CD") returns 3

See Also

BREAK

8.90 FIX Function

Returns the integer or whole part of a number.

FIX(*number***)**

Arguments

number
Number whose integer part will be returned.

Remarks

The **FIX** function is equivalent to the **IP** function and is provided for compatibility.

Examples

```
FIX(2.1)        returns 2
```

See Also

IP

8.91 FLOOR Function

Returns the floor of a number.

FLOOR(*number***)**

Arguments

number
Number whose floor will be returned.

Remarks

The **FLOOR** function returns the largest integer that is less than or equal to *number*.

The **FLOOR** function is equivalent to the **INT** function and is provided for compatibility.

Examples

```
FLOOR(1.8)      returns 1
FLOOR(-1.8)     returns -2
```

See Also

INT

8.92 FN Function

Calls a user-defined function and returns the result.

FN *function*[**(***argumentlist***)**]

Arguments

function
Name of function.

argumentlist
One or more arguments.

Remarks

The **FN** calls a function defined with the **DEF FN** statement. The number of arguments must match the number of parameters in the function definition.

Examples

```
DEF FN DOUBLE(X)=X*2
FN DOUBLE(2)          returns 4
```

See Also

DEF

8.93 FONT Function

Returns the current text font size.

FONT

8.94 FP Function

Returns the fractional part of a number.

FP(*number***)**

Arguments

number
Number whose fractional part will be returned.

Examples

```
FP(2.1)          returns 0.1
```

See Also

IP

8.95 FRE Function

Returns the amount of free memory or string space.

FRE(*expression***)**

Arguments

expression
Specifies which value to return.

Remarks

The **FRE** function returns the amount of free memory if the result of the expression is numeric or the amount of free string space if the result of the expression is a string.

The **FRE** function is equivalent to the **MEM** function and is provided for compatibility.

See Also

MEM

8.96 FREE Function

Returns the amount of free space on a volume.

FREE(*volume***)**

Arguments

volume
Volume whose free space will be returned.

Remarks

The **FREE** function returns the amount of free space in bytes on the specified volume.

8.97 FREEFILE Function

Returns the next free file number.

FREEFILE

8.98 FULLNAME$ Function

Returns the full file name of a file.

FULLNAME$(*filename*[**,***extension*]**)**

Arguments

filename
Name of file whose full file name will be returned.

extension
Extension to append to file if not included in file name.

Remarks

The **FULLNAME$** function creates a string containing the volume, file name and extension of a file. If the volume is not included in *filename* the default volume is used. If the extension is not included in *filename extension* is used.

If *extension* is omitted the default is "dat".

8.99 GET$ Function

Inputs a single character from the keyboard.

GET$

Remarks

The **GET$** function displays the cursor and waits for a character to be entered from the keyboard. The character is echoed to the screen and is immediately returned without requiring the ENTER key to be pressed.

See Also

GETALNUM$**, **GETALPHA$, **GETDIGIT$**, **GETYN$**

8.100 GETALNUM$ Function (standard version only)

Inputs a single alphanumeric character from the keyboard.

GETALNUM$

Remarks

The **GETALNUM$** function displays the cursor and waits for an alphanumeric character (a-z, A-Z, or 0-9) to be entered from the keyboard. The character is echoed to the screen and is immediately returned without requiring the ENTER key to be pressed.

Characters returned by the **GETALNUM$** function are upper case.

See Also

GET$**, **GETALPHA$, **GETDIGIT$**, **GETYN$**

8.101 GETALPHA$ Function (standard version only)

Inputs a single alphabetic character from the keyboard.

GETALPHA$

Remarks

The **GETALPHA$** function displays the cursor and waits for an alphabetic character (a-z or A-Z) to be entered from the keyboard. The character is echoed to the screen and is immediately returned without requiring the ENTER key to be pressed.

Characters returned by the **GETALPHA$** function are upper case.

See Also

GET$**, **GETALNUM$, **GETDIGIT$**, **GETYN$**

8.102 GETDIGIT$ Function (standard version only)

Inputs a single numeric character from the keyboard.

GETDIGIT$

Remarks

The **GETDIGIT$** function displays the cursor and waits for a numeric character (0-9) to be entered from the keyboard. The character is echoed to the screen and is immediately returned without requiring the ENTER key to be pressed.

See Also

GET$**, **GETALNUM$, **GETALPHA$**, **GETYN$**

8.103 GETYN$ Function (standard version only)

Inputs a single Y or N character from the keyboard.

GETYN$

Remarks

The **GETYN$** function displays the cursor and waits for a Y or N character to be entered from the keyboard. The character is echoed to the screen and is immediately returned without requiring the ENTER key to be pressed.

Characters returned by the **GETYN$** function are upper case.

See Also

GET$**, **GETALNUM$, **GETALPHA$**, **GETDIGIT$**

8.104 HEX Function

Determines if a character is a hexadecimal digit.

HEX(*character***)**

Arguments

character
Character to check.

Remarks

The **HEX** function checks if a character is a hexadecimal digit (A-F,a-f,0-9).

Examples

```
HEX("A")        returns -1 (true)
HEX("1")        returns -1 (true)
HEX("$")        returns 0 (false)
```

See Also

ALNUM, **ALPHA**, **CNTRL**, **DIGIT**, **HEX**, **LOWER**, **PRINT**, **UPPER**

8.105 HEX$ Function

Returns a string representing the hexadecimal value of a number.

HEX$(*number***)**

Arguments

number
Number whose hexadecimal string representation will be returned.

Examples

```
HEX$(35)        returns "23"
```

See Also

BIN$**, **OCT$, **STR$**

8.106 HOUR Function

Returns the hour from a time value.

HOUR(*time***)**

Arguments

time
Time value whose hour will be returned.

Examples

```
HOUR(MKTIME(2002,7,24,11,28,45))          returns 11
```

See Also

DAY, **MINUTE**, **MONTH**, **SECOND**, **WEEKDAY**, **YEAR**

8.107 HPOINT Function (Version 1.5 or later, standard version only)

Returns the color of a graphics pixel.

HPOINT(*x***,***y***)**

Arguments

x
X coordinate of pixel.

y
Y coordinate of pixel.

Remarks

Graphics must be turned on to get the color of a pixel.

The **HPOINT** function is equivalent to the **POINT** function and is provided for compatibility.

See Also

GRAPH, **HRESET**, **HSET**, **POINT**

8.108 HYPOT Function

Returns the hypotenuse.

HYPOT(*a***,***b***)**

Arguments

a,*b*

Length of sides of right triangle.

Remarks

The **HYPOT** function calculates the length of the hypotenuse of a right triangle given the length of the two sides *a* and *b* (the square root of $a^2 + b^2$).

Examples

```
HYPOT(3,4)        returns 5
```

8.109 IIF Function (standard version only)

Returns a number based on a condition.

IIF(*condition*,*number1*,*number2***)**

Arguments

condition
Condition to check.

number1
Number to return if condition is true.

number2
Number to return if condition is false.

Examples

```
IIF(1<=2,0,100)        returns 0
IIF("A"="B",1,2)       returns 2
```

See Also

IIF$

8.110 IIF$ Function (standard version only)

Returns a string based on a condition.

IIF$(*condition*,*string1*,*string2***)**

Arguments

condition
Condition to check.

string1
String to return if condition is true.

string2
String to return if condition is false.

Examples

```
IIF$(1<=2,"A","B")                          returns "A"
IIF$("A"="B","EQUAL","NOT EQUAL")           returns "NOT EQUAL"
```

See Also

IIF

8.111 INKEY$ Function

Returns a string corresponding to a key press.

INKEY$

Remarks

The **INKEY$** function does not wait for a key to be pressed. If no key has been pressed, an empty string ("") is returned.

See Also

INYN$**, **WAITKEY$

8.112 INPUT$ Function

Reads input from the keyboard or a file.

INPUT$(*length*[**,***file*]**)**

Arguments

length
Number of characters to read.

file
File number of open file.

Remarks

If *file* is omitted the default is the keyboard.

See Also

EDIT$, **INPUT**, **OPEN**

8.113 INSERT$ Function (standard version only)

Inserts a string into another string.

INSERT$(*string1***,***index***,***string2***)**

Arguments

string1
String in which second string will be inserted.

index
Index in first string where second string will be inserted.

string2
String to be inserted.

Examples

```
INSERT$("ABC123",4,"XYZ")          returns "ABCXYZ123"
```

See Also

DELETE$**, **REPLACE$

8.114 INSTR Function

Returns the position of the first occurrence of a string within another string.

INSTR([*start*,]*string1*,*string2***)**

Arguments

start
Index at which to start search.

string1
String in which to search.

string2
String whose position in the first string will be returned.

Remarks

If *string2* is not found 0 is returned. If *start* is omitted the search starts at the beginning of the string.

Examples

```
INSTR("A STRING","RING")           returns 5
INSTR("A STRING","CUP")            returns 0
INSTR(2,"TESTING","T")             returns 4
```

8.115 INSTRREV Function

Returns the position of the first occurrence of a string within another string from the end.

INSTRREV(*string1*,*string2*[,*start*]**)**

Arguments

string1
String in which to search.

string2
String whose position in the first string will be returned.

start
Index at which to start search.

Remarks

If *string2* is not found 0 is returned. If *start* is omitted the search starts at the end of the string.

Examples

```
INSTRREV("A STRING","RING")       returns 5
INSTRREV("A STRING","CUP")        returns 0
INSTRREV("TESTING","T",2)         returns 1
```

8.116 INT Function

Returns the floor of a number.

INT(*number***)**

Arguments

number
Number whose floor will be returned.

Remarks

The **INT** function returns the largest integer that is less than or equal to *number*.

Examples

```
INT(1.8)        returns 1
INT(-1.8)       returns -2
```

See Also

CEIL

8.117 INV Function

Returns the inverse of a number.

INV(*number***)**

Arguments

number
Number whose inverse will be returned.

Remarks

The **INV** function returns the inverse (1/*number*) of *number*.

Examples

```
INV(2)          returns 0.5
INV(0.5)        returns 2
```

8.118 INYN$ Function (standard version only)

Returns a string corresponding to a key press.

INYN$

Remarks

The **INYN$** waits for the Y or N keys to be pressed and returns "Y" or "N" respectively.

See Also

INKEY$**, **WAITKEY$

8.119 IP Function

Returns the integer or whole part of a number.

IP(*number***)**

Arguments

number
Number whose integer part will be returned.

Examples

```
IP(2.1)          returns 2
```

See Also

FP

8.120 IS0 Function (Version 1.1 or later)

Replaces 0 with the specified value.

IS0(*number1***,***number2***)**

Arguments

number1
Number to be checked.

number2
Number to return if *number1* is 0.

Remarks

The **IS0** function returns *number1* if it is not 0, otherwise returns *number2*.

Examples

```
IS0(5,1)         returns 5
IS0(0,1)         returns 1
```

See Also

ISEMPTY$, **ISNEG**, **ISPOS**

8.121 ISEMPTY$ Function (Version 1.1 or later)

Replaces an empty string with the specified value.

ISEMPTY$(*string1***,***string2***)**

Arguments

string1
String to be checked.

string2
String to return if *string1* is an empty string.

Remarks

The **ISEMPTY$** function returns *string1* if it is not an empty string, otherwise returns *string2*.

Examples

```
ISEMPTY$("ABC","123")        returns "ABC"
ISEMPTY$("","123")           returns "123"
```

See Also

IS0, **ISNEG**, **ISPOS**

8.122 ISNEG Function (Version 1.2 or later)

Replaces a negative value with the specified value.

ISNEG(*number1***,***number2***)**

Arguments

number1
Number to be checked.

number2
Number to return if *number1* is negative.

Remarks

The **ISNEG** function returns *number1* if it is not negative, otherwise returns *number2*.

Examples

```
ISNEG(-5,1)          returns 1
ISNEG(0,1)           returns 0
```

See Also

IS0, **ISEMPTY$**, **ISPOS**

8.123 ISPOS Function (Version 1.2 or later)

Replaces a positive value with the specified value.

ISPOS(*number1***,***number2***)**

Arguments

number1
Number to be checked.

number2
Number to return if *number1* is positive.

Remarks

The **ISPOS** function returns *number1* if it is not positive, otherwise returns *number2*.

Examples

```
ISPOS(5,1)          returns 1
ISPOS(0,1)          returns 1
```

See Also

IS0, **ISEMPTY$**, **ISNEG**

8.124 LBOUND Function (Version 1.2 or later)

Returns the lower bound (index) of a dimension in an array variable.

LBOUND(*array*[**,***dimension*]**)**

Arguments

array
Name of numeric or string array variable.

dimension
Dimension of array whose lower bound will be returned.

Remarks

The lower bound (base index) of a dimension depends on the **OPTION BASE** setting when the array is created.

If *dimension* is omitted the default is 1.

Examples

`LBOUND(A)` returns 0 (If **OPTION BASE** is not set or **OPTION BASE 0** is specified)

See Also

OPTION BASE, **MAXSIZE**, **SIZE**, **UBOUND**

8.125 LCASE$ Function

Converts a string to lower case.

LCASE$(*string***)**

Arguments

string
String to be converted to lower case.

Examples

```
LCASE$("ABC123")          returns "abc123"
```

See Also

UCASE$

8.126 LEAPYEAR Function

Determines if a year is a leap year.

LEAPYEAR(*year***)**

Arguments

year
Year to check.

Examples

```
LEAPYEAR(2002)      returns 0 (false)
LEAPYEAR(2000)      returns -1 (true)
```

8.127 LEFT$ Function

Returns the specified number of leftmost characters in a string.

LEFT$(*string***[,***length***])**

Arguments

string
String whose leftmost characters will be returned.

length
Number of characters to return.

Remarks

If *length* is omitted the default is 1.

Examples

```
LEFT$("ABC123")           returns "A"
LEFT$("ABC123",3)         returns "ABC"
```

See Also

MID$**, **RIGHT$

8.128 LEN Function

Returns the length of a string.

LEN(*string***)**

Arguments

string
String whose length will be returned.

Examples

```
LEN("HELLO")
```
returns 5

8.129 LOC Function

Returns the location of the file pointer of an open file.

LOC(*file***)**

Arguments

file
Number of file whose file pointer location will be returned.

Remarks

The **LOC** function returns the current record of an open random access file or the file pointer location of an open sequential file.

See Also

EOF, **LOF**, **REC**

8.130 LOF Function

Returns the length of an open file.

LOF(*file***)**

Arguments

file
Number of file whose length will be returned.

Remarks

The **LOF** function returns the length in bytes of an open file.

See Also

EOF, **LOC**, **REC**

8.131 LOG Function

Returns the natural logarithm of a number.

LOG(*number***)**

Arguments

number
Number whose natural logarithm will be returned.

Remarks

The **LOG** function returns the natural logarithm (*x*) of a number where *number*=e^x (*number* > 0).

Examples

```
LOG(7.38905609893068)
```
returns 2

See Also

LOG2, **LOG10**

8.132 LOG$ Function (standard version only)

Returns the file name of the log file if open.

LOG$

See Also

LOG

8.133 LOG2 Function

Returns the base 2 logarithm of a number.

LOG2(*number***)**

Arguments

number
Number whose base 2 logarithm will be returned.

Remarks

The **LOG2** function returns the base 2 logarithm (*x*) of a number where *number*=2^x (*number* > 0).

Examples

```
LOG2(4)
```
returns 2

See Also

LOG, **LOG10**

8.134 LOG10 Function

Returns the base 10 logarithm of a number.

LOG10(*number***)**

Arguments

number
Number whose base 10 logarithm will be returned.

Remarks

The **LOG10** function returns the base 10 logarithm (*x*) of a number where *number*=10^x (*number* > 0).

Examples

```
LOG10(100)
```
returns 2

See Also

LOG, **LOG2**

8.135 LOWER Function

Determines if a character is lowercase.

LOWER(*character***)**

Arguments

character
Character to check.

Examples

```
LOWER("A")
```
returns 0 (false)

```
LOWER("a")
```
returns -1 (true)

See Also

ALPHA, **CNTRL**, **DIGIT**, **HEX**, **PRINT**, **UPPER**

8.136 LOWER$ Function

Converts a string to lower case.

LOWER$(*string***)**

Arguments

string
String to be converted to lower case.

Remarks

The **LOWER$** function is equivalent to the **LCASE$** function and is provided for compatibility.

Examples

```
LOWER$("ABC123")          returns "abc123"
```

See Also

LCASE$**, **UPPER$

8.137 LPAD$ Function (standard version only)

Pads a string on the left.

LPAD$(*string***,***length***[,***pad***])**

Arguments

string
String which to pad.

length
Desired length of padded string.

pad
Character to pad string with.

Remarks

If *pad* is omitted the default is a space.

Examples

```
LPAD$("TEST",10)          returns "      TEST"
LPAD$("TEST",10,"*")      returns "******TEST"
```

See Also

RPAD$

8.138 LPOS Function (Version 1.1 or later, standard version only)

Returns the current print position.

LPOS

See Also

LPRINT, **LPRINT USING**, **OPEN PRINTER**

8.139 LSET$ Function (Version 1.4 or later, standard version only)

Left justifies a string.

LSET$(*string***,***length***[,***pad***])**

Arguments

string
String which to justify.

length
Desired length of justified string.

pad
Character to pad string with.

Remarks

If *pad* is omitted the default is a space.

The **LSET$** function is equivalent to the **LPAD$** function and is provided for compatibility.

Examples

```
LSET$("TEST",10)          returns "     TEST"
LSET$("TEST",10,"*")      returns "******TEST"
```

See Also

LPAD$**, **RSET$

8.140 LTRIM$ Function

Trims spaces from the left of a string.

LTRIM$(*string***)**

- or -

LTRIM$(*string*[**,***character*]**)** (Version 1.4 or later)

Arguments

string
String to be trimmed.

character
Character to trim.

Remarks

If *character* is omitted the default is a space.

Examples

```
LTRIM$("   ABC   ")          returns "abc  "
LTRIM$("##ABC##","#")       returns "abc##"
```

See Also

COMPRESS$**, **RTRIM$, **TRIM$**

8.141 MAKENAME$ Function

Makes a filename.

MAKENAME$([*volume*],*name*[,[*extension*][,*replace*]]**)**

Arguments

volume
Volume.

name
Filename.

extension
Extension.

replace
Whether or not to replace the volume and extension.

Remarks

The **MAKENAME$** function combines a volume, file name and extension into a file specification. If *replace* is 1 (TRUE), the volume and extension will be replaced in the *name* if present with the specified *volume* and *extension*.

If *volume* is omitted the default is the default volume. If *extension* is omitted the default is "DAT". If *replace* is omitted the default is 0 (FALSE).

Examples

```
MAKENAME$("HOME","TEST")            returns "HOME:TEST.DAT"
MAKENAME$("WORK","ACCTS","REG")     returns "WORK:ACCTS.REG"
MAKENAME$("EXAMPLES","DATA:LIST.DAT","WRK",1)        returns
                                    "EXAMPLES:LIST.WRK"
```

See Also

SPLITNAME, **SPLITNAME$**

8.142 MAPPED Function

Determines if a volume is mapped (exists).

MAPPED(*volume***)**

Arguments

volume
Volume to check.

Examples

```
MAPPED("EXAMPLES")      returns -1 (true) if EXAMPLES is mapped, otherwise
                        returns 0 (false).
```

See Also

DEFAULT, **READONLY**

8.143 MATCH Function (Version 1.5 or later)

Returns the position of the first occurrence of a string within another string.

MATCH([*start*,]*string1*,*string2***)**

Arguments

start
Index at which to start search.

string1
String in which to search.

string2
String whose position in the first string will be returned.

Remarks

If *string2* is not found 0 is returned. If *start* is omitted the search starts at the beginning of the string.

The **MATCH** function is equivalent to the **INSTR** function and is provided for compatibility.

Examples

```
MATCH("A STRING","RING")          returns 5
MATCH("A STRING","CUP")           returns 0
MATCH(2,"TESTING","T")            returns 4
```

See Also

INSTR

8.144 MAX Function

Returns the maximum of two or more numbers.

MAX(*number1*,*number2***)**

- or -

MAX(*number*[,*number*]...**)** (Version 1.4 or later)

Arguments

number1
First number to check.

number2
Second number to check.

number
Number to check.

Examples

```
MAX(5,1)        returns 5
MAX(5,1,2)      returns 5
```

See Also

MIN

8.145 MAXLEN Function

Returns the maximum length of a string variable.

MAXLEN(*variable***)**

Arguments

variable
String variable whose maximum length will be returned.

Remarks

The **MAXLEN** function can also be used to get the width of a field variable used in a **FIELD** statement.

Examples

```
MAXLEN(A$)      returns 65535
```

See Also

FIELD

8.146 MAXNUM Function

Returns the largest finite positive number.

MAXNUM

See Also

MINNUM

8.147 MAXSIZE Function (Version 1.2 or later)

Returns the total number of elements of all dimensions in an array variable.

MAXSIZE(*array***)**

Arguments

array
Name of numeric or string array variable.

Remarks

The number of elements in an array is specified when the array is created either automatically or using the **DIM** statement and depends on the **OPTION BASE** setting.

Examples

`MAXSIZE(A)`	returns 11 (If the array A is created either automatically or using `DIM A(10)` and **OPTION BASE** is not set or **OPTION BASE 0** is specified)

See Also

DIM, **OPTION BASE**, **LBOUND**, **SIZE**, **UBOUND**

8.148 MEM Function

Returns the amount of free memory.

MEM

8.149 MID$ Function

Returns part of a string.

MID$(*string***,***index*[**,***length*]**)**

Arguments

string
String from which substring will be returned.

index
Start of substring to return.

length
Number of characters in substring to return.

Remarks

If *length* is omitted the default is 1.

Examples

`MID$("ABC123",3)`	returns "C"
`MID$("ABC123",3,2)`	returns "C1"

See Also

LEFT$**, **RIGHT$

8.150 MIN Function

Returns the minimum of two or more numbers.

MIN(*number1***,***number2***)**

- or -

MIN(*number*[**,***number*]...**)** (Version 1.4 or later)

Arguments

number1
First number to check.

number2
Second number to check.

number
Number to check.

Examples

```
MIN(5,1)        returns 1
MIN(5,1,2)      returns 1
```

See Also

MAX

8.151 MINNUM Function

Returns the smallest finite positive number.

MINNUM

See Also

MAXNUM

8.152 MINUTE Function

Returns the minute from a time value.

MINUTE(*time***)**

Arguments

time
Time value whose minute will be returned.

Examples

```
MINUTE(MKTIME(2002,7,24,11,28,45))        returns 28
```

See Also

DAY, **HOUR**, **MONTH**, **SECOND**, **WEEKDAY**, **YEAR**

8.153 MKKEY$ Function

Creates a string representing a key.

MKKEY$(*key***[,[***control***][,***shift***]])**

Arguments

key
Key whose string representation will be returned.

control
Specifies if CTRL key is pressed.

shift
Specifies if SHIFT key is pressed.

Remarks

The **MKKEY$** function does not perform any key translations. For example MKKEY$("A",1) returns CTRL+A not CTRLA (1).

If *control* is omitted the default is false (0). If *shift* is omitted the default is false (0).

Examples

`MKKEY$(CHR$(0)+CHR$(35),1)`	returns a string representing CTRL+END
`MKKEY$(CHR$(0)+CHR$(35),,1)`	returns a string representing SHIFT+END

8.154 MKN$ Function

Converts a number to a binary string.

MKN$(*number***)**

Arguments

number
Number to convert.

Remarks

The **MKN$** function converts a number to a binary string to be assigned to a field variable.

See Also

CVN

8.155 MKTIME Function

Creates a time value.

MKTIME(*year***,***month***,***day***[,***hour***[,***minute***[,***second***]]])**

Arguments

year
Year for time value.

month
Month for time value.

day
Day of the month for time value.

hour
Hour for time value.

minute
Minute for time value.

second
Second for time value.

Remarks

The **MKTIME** function creates a time value from individual date and time values.

If all the time values (*hour*, *minute*, and *second*) are omitted the default time is 00:00:00 AM. If *minute* is omitted the default is 0. If *second* is omitted the default is 0.

Examples

`MKTIME(2002,7,24)`	returns 12671942400
`MKTIME(2002,7,24,11,28,45)`	returns 12671983725

8.156 MOD Function

Returns the modulus of two numbers.

MOD(*number1***,***number2***)**

Arguments

number1
Dividend.

number2
Divisor.

Remarks

The **MOD** function divides one number by another number and returns the remainder. The numbers are rounded to integers before dividing.

Examples

`MOD(5,2)`	returns 1
`MOD(9.8,2.1)`	returns 0

See Also

MOD (operator)

8.157 MONTH Function

Returns the month from a time value.

MONTH(*time***)**

Arguments

time
Time value whose month will be returned.

Examples

```
MONTH(MKTIME(2002,7,24))     returns 7
```

See Also

DAY, **HOUR**, **MINUTE**, **SECOND**, **WEEKDAY**, **YEAR**

8.158 MONTHNAME$ Function

Returns the month name from a time value.

MONTHNAME$(*month*[**,***format*]**)**

Arguments

month
Month of year whose name will be returned.

format
Format of day name.

Remarks

If *format* is omitted the long name of the month is returned. If *format* is non-zero the short name of the month is returned.

Examples

```
MONTHNAME$(1)          returns "January"
MONTHNAME$(1,1)        returns "Jan"
```

See Also

DAYNAME$

8.159 NOW Function

Returns the current time value.

NOW

See Also

ADJUST

8.160 NUL$ Function

Returns the NULL character (decimal 0).

NUL$

8.161 OCT$ Function

Returns a string representing the octal value of a number.

OCT$(*number***)**

Arguments

number
Number whose octal string representation will be returned.

Examples

```
OCT$(35)        returns "43"
```

See Also

BIN$**, **HEX$, **STR$**

8.162 ODD Function (standard version only)

Determines if a number is odd.

ODD(*number***)**

Arguments

number
Number to check.

Examples

```
ODD(2)          returns 0 (false)
ODD(3)          returns -1 (true)
```

See Also

EVEN

8.163 OPEN Function

Determines if a file or the printer is open.

OPEN(*file***)**

Arguments

file
Number of file or printer (-2) to check.

Remarks

Use **OPEN(-2)** to check if the printer is open.

Examples

```
OPEN(1)     returns -1 if the file is open, 0 otherwise.
```

See Also

OPEN

8.164 ORD Function

Returns the ordinal value of a character.

ORD(*character***)**

Arguments

character
Character whose ordinal value will be returned.

Examples

```
ORD("A")        returns 65
ORD("1")        returns 49
ORD("$")        returns 36
```

See Also

CHR$

8.165 PCOL Function (Version 1.5.2 or later, standard version only)

Returns the graphics coordinate of a text column.

PCOL(*column***)**

Arguments

column
Text column.

Remarks

The **PCOL** function returns the x coordinate on the graphics screen of a column on the text screen.

Graphics must be turned on to get graphics coordinate of a column.

See Also

PROW

8.166 PEEK Function

Returns the value at a memory location.

PEEK(*address***)**

Arguments

address
Address of memory location whose value will be returned.

Examples

```
PEEK(39038)
```
returns 8 (default tab size)

See Also

EXEC, **POKE**

8.167 PERM Function

Returns the number of permutations.

PERM(*n*,*k***)**

Arguments

n
Number of objects.

k
Number of objects in each set.

Remarks

The **PERM** function calculates the number of permutations of *n* objects taken *k* at a time (*n*!/(*n*-*k*)!).

Examples

```
PERM(5,2)
```
returns 20

See Also

COMB

8.168 PFONT Function (standard version only)

Returns the current graphics font size.

PFONT

Remarks

Graphics must be turned on to get the graphics font size.

8.169 PI Function

Returns the value of Pi (π).

PI

8.170 POINT Function (standard version only)

Returns the color of a graphics pixel.

POINT(*x*,*y***)**

Arguments

x
X coordinate of pixel.

y
Y coordinate of pixel.

Remarks

Graphics must be turned on to get the color of a pixel.

See Also

GRAPH, **PRESET**, **PSET**

8.171 POS Function

Returns the current position of the cursor.

POS[**(***position***)**]

Arguments

position
Specifies which position of the cursor to return.

Remarks

If *position* is greater than or equal to 0, the current column is returned, otherwise the current row is returned.

If *position* is omitted, the current column is returned.

The **POS** function is equivalent to the **COLUMN** and **ROW** functions and is provided for compatibility.

Examples

`POS`	returns the current column position of the cursor
`POS(-1)`	returns the current row position of the cursor

See Also

COLUMN, **ROW**

8.172 PPOINT Function (standard version only)

Returns the color of a graphics pixel.

PPOINT(*x***,***y***)**

Arguments

x
X coordinate of pixel.

y
Y coordinate of pixel.

Remarks

Graphics must be turned on to get the color of a pixel.

The **PPOINT** function is equivalent to the **POINT** function and is provided for compatibility.

See Also

POINT

8.173 PRINT Function

Determines if a character is a printable character.

PRINT(*character***)**

Arguments

character
Character to check.

Remarks

The **PRINT** function checks if a character is a printable character (&H20-&H7E).

Examples

```
PRINT("A")            returns -1 (true)
PRINT("1")            returns -1 (true)
PRINT(CHR$(&H10))     returns 0 (false)
```

See Also

ALNUM, **ALPHA**, **CNTRL**, **DIGIT**, **HEX**, **LOWER**, **UPPER**

8.174 PRINTER$ Function (Version 1.1 or later, standard version only)

Returns the name of the current printer.

PRINTER$

Remarks

The **PRINTER$** function returns the name of the most recent printer set using the **PRINTER** statement or if not set, the name of the printer as specified in the Print Setup or Print dialogs.

See Also

OPEN PRINTER

8.175 PROMPT$ Function (standard version only)

Returns the prompt.

PROMPT$

8.176 PROPER$ Function (standard version only)

Capitalizes a string.

PROPER$(*string***)**

Arguments

string
String to be capitalized.

Remarks

The **PROPER$** function capitalizes the first character in the string and every character that does not follow a letter.

Examples

```
PROPER$("abc123def")
```
returns "Abc123Def"

8.177 PROW Function (standard version only)

Returns the graphics coordinate of a text row.

PROW(*row***)**

Arguments

row
Text row.

Remarks

The **PROW** function returns the y coordinate on the graphics screen of a row on the text screen.

Graphics must be turned on to get graphics coordinate of a row.

See Also

PCOL

8.178 PSCRH Function (standard version only)

Returns the height (pixels) of the graphics screen.

PSCRH

Remarks

Graphics must be turned on to get the graphics screen height.

See Also

PSCRW

8.179 PSCRW Function (standard version only)

Returns the width (pixels) of the graphics screen.

PSCRW

Remarks

Graphics must be turned on to get the graphics screen width.

See Also

PSCRH

8.180 QUOTE$ Function

Quotes a string.

QUOTE$(*string*[**,***quotes*]**)**

Arguments

string
String to be quoted.

quotes
Characters to use as quotes (2).

Remarks

The **QUOTE$** function encloses a string with the quote characters specified.

if *quotes* is omitted the default is double quotes (""").

Examples

```
QUOTE$("hello")          returns ""hello""
QUOTE$("abcd","[]")      returns "[abcd]"
```

8.181 RAD Function

Converts degrees to radians.

RAD(*degrees***)**

Arguments

degrees
Degrees whose value in radians will be returned.

Examples

```
RAD(180)          returns 3.14159265358979
```

See Also

DEG

8.182 RCP Function

Returns the inverse of a number.

RCP(*number***)**

Arguments

number
Number whose inverse will be returned.

Remarks

The **RCP** function returns the inverse (1/*number*) of *number*.

The **RCP** function is equivalent to the **INV** function and is provided for compatibility.

Examples

`RCP(2)`	returns 0.5
`RCP(0.5)`	returns 2

See Also

INV

8.183 READONLY Function

Determines if a volume is read-only.

READONLY(*volume***)**

Arguments

volume
Volume to check.

Examples

`READONLY("EXAMPLES")`	returns -1 (true) if EXAMPLES is read-only, otherwise returns 0 (false).

See Also

DEFAULT, **MAPPED**

8.184 REC Function

Returns the number of records in an open random access file.

REC(*file***)**

Arguments

file
Number of file whose record count will be returned.

See Also

EOF, **LOC**, **LOF**

8.185 REMAIN$ Function (standard version only)

Returns part of a string after the first occurrence of another string.

REMAIN$(*string1***,***string2***)**

Arguments

string1
String from which a part will be returned.

string2
String before which the part to be returned begins.

Examples

REMAIN$("ABCDEF","CD") returns "EF"

See Also

EXTRACT$

8.186 REMAINDER Function

Returns the remainder of dividing one number by another number.

REMAINDER(*number1***,***number2***)**

Arguments

number1
Number to be divided.

number2
Number to divide by.

Remarks

The **REMAINDER** function divides *number1* by *number2* and returns the remainder after a non-fractional number of divisions.

Examples

`REMAINDER(11.5,1.25)` returns 0.25

8.187 REMOVE$ Function (standard version only)

Removes each occurrence of a string from another string.

REMOVE$(*string1***,***string2***)**

Arguments

string1
String from which to remove a string.

string2
String to remove.

Examples

```
REMOVE$("ABC123","ABC")
```
returns "123"

See Also

CHANGE$

8.188 REPEAT$ Function

Returns a string repeated a specified number of times.

REPEAT$(*string***,***repeat***)**

Arguments

string
String to be repeated.

repeat
Number of times to repeat the string.

Examples

```
REPEAT$("ABC",2)
```
returns "ABCABC"

See Also

STRING$

8.189 REPLACE$ Function (standard version only)

Replaces part of a string with another string.

REPLACE$(*string1***,***index***,***string2***)**

Arguments

string1
String in which replacement will occur.

index
Index in first string where replacement will begin.

string2
Replacement string.

Examples

```
REPLACE$("ABC123",4,"XYZ")          returns "ABCXYZ"
```

See Also

DELETE$**, **REPLACE$

8.190 RET$ Function

Returns the CARRIAGE RETURN character (decimal 13).

RET$

8.191 REVERSE$ Function

Reverses a string.

REVERSE$(*string***)**

Arguments

string
String to be reversed.

Examples

```
REVERSE$("ABC123")          returns "321CBA"
```

8.192 RFIND Function (standard version only)

Finds the position of a character within a string.

RFIND(*string***,***character*[**,***index*]**)**

Arguments

string
String in which to search.

character
Character to search for.

index
Position where search is to begin.

Remarks

The **RFIND** function performs a reverse search from the end of the string (or *index* if specified) to the beginning.

If *character* is not found 0 is returned.

If *index* is omitted the default is the end of the string.

Examples

```
RFIND("A STRING","R")          returns 5
RFIND("A STRING","R",3)        returns 0
```

See Also

FIND

8.193 RIGHT$ Function

Returns the specified number of rightmost characters in a string.

RIGHT$(*string*[**,***length*]**)**

Arguments

string
String whose rightmost characters will be returned.

length
Number of characters to return.

Remarks

If *length* is omitted the default is 1.

Examples

```
RIGHT$("ABC123")          returns "3"
RIGHT$("ABC123",3)        returns "123"
```

See Also

LEFT$**, **MID$

8.194 RND Function

Returns a random number.

RND(*number***)**

Arguments

number
Specifies the range of the random number to be returned.

Remarks

The **RND** function returns a random number between 0 and 1 if *number* is 0 or 1 or from 1 to *number* if *number* is > 1.

Examples

```
RND(0)    returns a random number between 0 and 1.
RND(1)    returns a random number between 0 and 1.
RND(10)   returns a random number between 1 and 10.
```

8.195 ROUND Function

Rounds a number.

ROUND(*number*,*digits*)**

Arguments

number
Number to be rounded.

digits
Number of digits after the decimal point.

Examples

```
ROUND(1.23456,0)      returns 1
ROUND(1.23456,2)      returns 1.23
ROUND(1.23456,3)      returns 1.235
```

See Also

TRUNCATE

8.196 ROW Function

Returns the row position of the cursor.

ROW

See Also

COLUMN

8.197 RPAD$ Function (standard version only)

Pads a string on the right.

RPAD$(*string*,*length*[,*pad*]**)**

Arguments

string
String which to pad.

length
Desired length of padded string.

pad
Character to pad string with.

Remarks

If *pad* is omitted the default is a space.

Examples

```
RPAD$("TEST",10)            returns "TEST      "
RPAD$("TEST",10,"*")        returns "TEST******"
```

See Also

LPAD$

8.198 RSET$ Function (Version 1.4 or later, standard version only)

Right justifies a string.

RSET$(*string*,*length*[,*pad*]**)**

Arguments

string
String which to justify.

length
Desired length of justified string.

pad
Character to pad string with.

Remarks

If *pad* is omitted the default is a space.

The **RSET$** function is equivalent to the **RPAD$** function and is provided for compatibility.

Examples

```
RSET$("TEST",10)          returns "TEST      "
RSET$("TEST",10,"*")      returns "TEST******"
```

See Also

LSET$**, **RPAD$

8.199 RTD Function

Converts radians to degrees.

RTD(*radians***)**

Arguments

radians
Radians whose value in degrees will be returned.

Remarks

The **RTD** function is equivalent to the **DEG** function and is provided for compatibility.

Examples

```
RTD(PI)   returns 180
```

See Also

DEG

8.200 RTRIM$ Function

Trims spaces from the right of a string.

RTRIM$(*string***)**

- or -

RTRIM$(*string*[**,***character*]**)** (Version 1.4 or later)

Arguments

string
String to be trimmed.

character
Character to trim.

Remarks

If *character* is omitted the default is a space.

Examples

```
RTRIM$("   ABC   ")            returns "  abc"
RTRIM$("##ABC##","#")          returns "##abc"
```

See Also

COMPRESS$**, **LTRIM$, **TRIM$**

8.201 SADD Function (Version 1.5 or later)

Returns the address of a string variable.

SADD(*variable***)**

Arguments

variable
The string variable whose address is to be returned.

Remarks

The **SADD** function returns a number that identifies the location in memory of the specified variable.

The **SADD** function is similar to the **VARPTR** function and is provided for compatibility.

Examples

```
SADD(NM$)          returns the address of the string variable NM$
```

See Also

VARPTR

8.202 SCREEN Function (Version 1.4 or later)

Saves the screen.

SCREEN(*row***,***column***,***columns***,***rows*[**,**[*foreground*][**,***background*]]**)**

Parameters

row
Starting row.

column
Starting column.

columns
Number of columns to fill.

rows
Number of rows to fill.

foreground
Foreground color of saved screen.

background
Background color of saved screen.

Remarks

The **SCREEN** function saves a specified number of rows and columns of the screen beginning at *row*,*column*, sets the foreground and background colors to the colors specified and clears the saved part of the screen. Up to 255 screens can be saved. Use the **SCREEN RESTORE** statement to restore a screen.

The **SCREEN** function returns a number identifying the saved screen, which can be used in the **SCREEN RESTORE** statement to restore the saved screen.

If *foreground* is omitted the current foreground text color is used. If *background* is omitted the current background text color is used.

Examples

```
SCREEN(20,5,40,15)          returns screen id
SCREEN(20,5,40,15,7)        returns screen id
SCREEN(20,5,40,15,7,10)     returns screen id
SCREEN(20,5,40,15,,10)      returns screen id
```

Mode

Program only

See Also

SCREEN BACKUP, **SCREEN RESTORE**

8.203 SCRH Function

Returns the height (rows) of the screen.

SCRH

See Also

SCRW

8.204 SCRN Function (Version 1.4 or later, standard version only)

Returns the color of a graphics pixel.

SCRN(*x*,*y***)**

Arguments

x
X coordinate of pixel.

y
Y coordinate of pixel.

Remarks

Graphics must be turned on to get the color of a pixel.

The **SCRN** function is equivalent to the **POINT** function and is provided for compatibility.

See Also

POINT

8.205 SCRW Function

Returns the width (columns) of the screen.

SCRW

See Also

SCRH

8.206 SEC Function

Returns the secant of an angle.

SEC(*angle***)**

Arguments

angle
Angle in radians whose secant will be returned.

Remarks

The **SEC** function returns the secant (1/cos(*angle*)) of *angle*.

Examples

```
SEC(0)      returns 1
```

See Also

CSC

8.207 SECOND Function

Returns the second from a time value.

SECOND(*time***)**

Arguments

time
Time value whose second will be returned.

Examples

```
SECOND(MKTIME(2002,7,24,11,28,45))        returns 45
```

See Also

DAY, **HOUR**, **MINUTE**, **MONTH**, **WEEKDAY**, **YEAR**

8.208 SEG$ Function

Returns part of a string.

SEG$(*string***,***index***[,***length***])**

Arguments

string
String from which substring will be returned.

index
Start of substring to return.

length
Number of characters in substring to return.

Remarks

The **SEG$** function is equivalent to the **MID$** function and is provided for compatibility.

Examples

```
SEG$("ABC123",3)          returns "C"
SEG$("ABC123",3,2)        returns "C1"
```

See Also

MID$

8.209 SELECT Function (Version 1.3 or later)

Changes the character set.

SELECT(*character-set***)**

Arguments

character-set
Character set to select.

Remarks

The **SELECT** function changes the current character set used to display text on the screen.

To select a character set, specify one of the following values in the **SELECT** function.

0	ANSI
161	Greek
162	Turkish
177	Hebrew
178	Arabic
186	Baltic
204	Russian
238	East Europe
255	OEM

To make sure the character set was successfully changed, compare the value returned by the **SELECT** function to the character set specified in the function call (e.g. `IF SELECT(255)=255 THEN` ...)

On startup, the character set is set to ANSI (0).

Examples

`SELECT(255)`	returns 255 if the character set was changed, otherwise returns the previous character-set

See Also

SELECT

8.210 SET$ Function

Sets characters in a string to a character.

SET$(*string1***,***string2***)**

Arguments

string1
String to be set.

string2
Character to set string to.

Remarks

The **SET$** function sets all the characters in a string to the specified character.

Examples

```
SET$("1234","*")          returns "****"
```

8.211 SGN Function

Returns the sign of a number.

SGN(*number***)**

Arguments

number
Number whose sign will be returned.

Examples

```
SGN(1)          returns 1
SGN(0)          returns 0
SGN(-1)         returns -1
```

See Also

ABS

8.212 SHIFT$ Function (Version 1.2 or later)

Returns the SHIFT key.

SHIFT$

Remarks

The **SHIFT$** function returns a string containing the character codes for the SHIFT key (&H00, &H10).

8.213 SIN Function

Returns the sine of an angle.

SIN(*angle***)**

Arguments

angle
Angle in radians whose sine will be returned.

Examples

```
SIN(0)          returns 0
SIN(PI/4)       returns 0.707106781186548
```

See Also

ASIN, **COS**, **SINH**

8.214 SINH Function

Returns the hyperbolic sine of a number.

SINH(*angle***)**

Arguments

angle
Angle in radians whose hyperbolic sine will be returned.

Examples

`SINH(PI)` returns 11.5487393572577

See Also

COSH, **SIN**

8.215 SIZE Function (Version 1.2 or later)

Returns the number of elements of a dimension in an array variable.

SIZE(*array*[**,***dimension*]**)**

Arguments

array
Name of numeric or string array variable.

dimension
Dimension of array whose number of elements will be returned.

Remarks

The number of elements in a dimension is specified when the array is created either automatically or using the **DIM** statement and depends on the **OPTION BASE** setting.

If *dimension* is omitted the default is 1.

Examples

`SIZE(A)` returns 11 (If the array A is created either automatically or using `DIM A(10)` and **OPTION BASE** is not set or **OPTION BASE 0** is specified)

See Also

DIM, **OPTION BASE**, **LBOUND**, **MAXSIZE**, **UBOUND**

8.216 SPACE$ Function

Returns a string containing a specified number of spaces.

SPACE$(*spaces***)**

Arguments

spaces
Number of spaces in string.

Examples

```
SPACE$(10)              returns "          "
```

8.217 SPAN Function (standard version only)

Returns the length of a substring beginning a string containing characters in another string.

SPAN(*string1***,***string2***)**

Arguments

string1
String to search.

string2
Characters to search for.

Examples

SPAN("ABABCDCD","ABC") returns 5

See Also

CSPAN

8.218 SPLITNAME$ Function (Version 1.1 or later)

Returns part of a file name.

SPLITNAME$(*filename*[**,***part*]**)**

Arguments

filename
Name of file.

part
Part of filename to return.

Remarks

The **SPLITNAME$** function returns the volume if *part* is -1, the name if *part* is 0 or the extension if *part* is 1.

If *part* is omitted the default is 0 (volume).

Examples

```
SPLITNAME$("EXAMPLES:TEST.DAT")        returns "TEST"
```

```
SPLITNAME$("EXAMPLES:TEST.DAT",-1)     returns "EXAMPLES"
SPLITNAME$("EXAMPLES:TEST.DAT",0)      returns "TEST"
SPLITNAME$("EXAMPLES:TEST.DAT",1)      returns "DAT"
```

See Also

SPLITNAME, **MAKENAME$**

8.219 SQR Function

Returns the square root of a number.

SQR(*number***)**

Arguments

number
Number whose square root will be returned.

Examples

`SQR(9)` returns 3

See Also

CBR

8.220 SQRT Function

Returns the square root of a number.

SQRT(*number***)**

Arguments

number
Number whose square root will be returned.

Remarks

The **SQRT** function is equivalent to the **SQR** function and is provided for compatibility.

Examples

`SQRT(9)` returns 3

See Also

SQR

8.221 SQUEEZE$ Function (Version 1.4 or later)

Trims spaces from the left and right of a string and compresses duplicate spaces to a single space.

SQUEEZE$(*string***)**

Arguments

string
String to be squeezed.

Examples

```
SQUEEZE$(" A      B C ")        returns "A B C"
```

Remarks

The **SQUEEZE$** function is equivalent to the **COMPRESS$** function and is provided for compatibility.

See Also

COMPRESS$

8.222 STR$ Function

Returns a string representing the decimal value of a number.

STR$(*number***)**

Arguments

number
Number whose decimal string representation will be returned.

Examples

```
STR$(35)          returns "35"
```

See Also

BIN$**, **HEX$, **OCT$**

8.223 STRING$ Function

Returns a string of a specified number of characters.

STRING$(*character***,***length***)**

Arguments

character
Character to repeat.

length
Number of characters to return.

Examples

```
STRING$("A",4)          returns "AAAA"
```

See Also

REPEAT$

8.224 SWITCH Function (standard version only)

Selects a number.

SWITCH(*condition***,***number*[**,***condition***,***number*]...**)**

Arguments

condition
Determines what number to select.

number
Number to select if *condition* is true.

Remarks

The **SWITCH** function conditionally selects a *number* from a list. The first *condition* that evaluates to true (-1) in the list specifies the *number* to select. At least one *condition* must evaluate to true (-1). The conditions are evaluated in the order that they appear in the list.

To provide a default value, include at the end of the list the values **TRUE** (-1) and the default *number* to select.

Examples

```
SWITCH(1>2,10,5=5,20,"A"<"B",30)          returns 20
SWITCH(1>2,10,5<>5,20,"A">"B",30,TRUE,0)  returns 0
```

See Also

CHOOSE, **CHOOSE$**, **SWITCH$**

8.225 SWITCH$ Function (standard version only)

Selects a string.

SWITCH$(*condition***,***string*[**,***condition***,***string*]...**)**

Arguments

condition
Determines what string to select.

string
String to select if *condition* is true.

Remarks

The **SWITCH$** function conditionally selects a *string* from a list. The first *condition* that evaluates to true (-1) in the list specifies the *string* to select. At least one *condition* must evaluate to true (-1). The conditions are evaluated in the order that they appear in the list.

To provide a default value, include at the end of the list the values **TRUE** (-1) and the default *string* to select.

Examples

```
SWITCH$(1>2,"A",5=5,"B","A"<"B","C")                returns "B"
SWITCH$(1>2,"A",5<>5,"B","A">"B","C",TRUE,"")       returns ""
```

See Also

CHOOSE, **CHOOSE$**, **SWITCH**

8.226 TAN Function

Returns the tangent of an angle.

TAN(*angle***)**

Arguments

angle
Angle in radians whose tangent will be returned.

Examples

```
TAN(PI/4)
```
returns 0.999999999999999

See Also

COT, **ATN**

8.227 TANH Function

Returns the hyperbolic tangent of an angle.

TANH(*angle***)**

Arguments

angle
Angle in radians whose hyperbolic tangent will be returned.

Examples

```
TANH(PI)
```
returns 0.99627207622075

See Also

TAN

8.228 TEMPNAME$ Function

Returns a unique temporary file name for a specified volume.

TEMPNAME$(*volume***,***extension***)**

Arguments

volume
Volume on which to check for a unique temporary file name.

extension

Extension of temporary file name.

Remarks

The **TEMPNAME$** function creates a unique file name for the specified volume and should be used for temporary files which will be deleted or renamed. The file names returned are of the form "ddddddddd.*ext*" where d is a numeric digit (0-9).

If *extension* is omitted the default is "tmp".

Examples

`TEMPNAME$("HOME")`	returns "HOME:00000001.tmp"
`TEMPNAME$("HOME","DAT")`	returns "HOME:00000002.DAT"

8.229 TIME Function

Returns the number of seconds since midnight.

TIME

8.230 TIME$ Function

Returns the current time as a string in the form HH:MM:SS.

TIME$[**(***time***)**]

Arguments

time
Time value whose time will be returned.

Remarks

If *time* is omitted the current time is returned.

Examples

`TIME$(MKTIME(2002,7,24,11,28,45))` returns "11:28:45"

See Also

DATE$

8.231 TIMER Function

Returns the value of the timer.

TIMER

8.232 TRIM$ Function

Trims spaces from the left and right of a string.

TRIM$(*string***)**

- or -

TRIM$(*string*[**,***character*]**)** (Version 1.4 or later)

Arguments

string
String to be trimmed.

character
Character to trim.

Remarks

If *character* is omitted the default is a space.

Examples

```
TRIM$("   ABC   ")              returns "abc"
TRIM$("##ABC##","#")           returns "abc"
```

See Also

COMPRESS$**, **LTRIM$, **RTRIM$**

8.233 TRUE Function

Returns true (-1).

TRUE

See Also

FALSE

8.234 TRUNCATE Function

Truncates a number.

TRUNCATE(*number***,***digits***)**

Arguments

number
Number to be truncated.

digits
Number of digits after the decimal point.

Examples

```
TRUNCATE(1.23456,0)   returns 1
TRUNCATE(1.23456,2)   returns 1.23
TRUNCATE(1.23456,3)   returns 1.234
```

See Also

ROUND

8.235 TWOPI Function

Returns the value of 2 * Pi (π).

TWOPI

8.236 UBOUND Function (Version 1.2 or later)

Returns the upper bound (index) of a dimension in an array variable.

UBOUND(*array*[**,***dimension*]**)**

Arguments

array
Name of numeric or string array variable.

dimension
Dimension of array whose upper bound will be returned.

Remarks

The upper bound of a dimension is specified when the array is created either automatically or using the **DIM** statement.

If *dimension* is omitted the default is 1.

Examples

`UBOUND(A)` returns 10 (If the array A is created either automatically or using `DIM A(10)`)

See Also

DIM, **OPTION BASE**, **LBOUND**, **MAXSIZE**, **SIZE**

8.237 UCASE$ Function

Converts a string to upper case.

UCASE$(*string***)**

Arguments

string
String to be converted to upper case.

Examples

`UCASE$("abc123")` returns "ABC123"

See Also

LCASE$

8.238 UPPER Function

Determines if a character is uppercase.

UPPER(*character***)**

Arguments

character
Character to check.

Examples

```
UPPER("A")      returns -1 (true)
UPPER("a")      returns 0 (false)
```

See Also

ALPHA, **CNTRL**, **DIGIT**, **HEX**, **LOWER**, **PRINT**

8.239 UPPER$ Function

Converts a string to upper case.

UPPER$(*string***)**

Arguments

string
String to be converted to upper case.

Remarks

The **UPPER$** function is equivalent to the **UCASE$** function and is provided for compatibility.

Examples

```
UPPER$("abc123")            returns "ABC123"
```

See Also

LOWER$**, **UCASE$

8.240 USING$ Function

Formats values.

USING$(*format***,***expression***[,***expression***]...)**

Arguments

format
Format string.

expression
Data to format.

Remarks

The **USING$** function returns a string containing the formatted values.

The format string consists of characters that specify how numbers and strings are formatted.

Numeric format characters:

#	Digit position.
.	Decimal point position.
,	Placed left of the decimal point, prints a comma in every third position.
+	Sign position.
-	Placed after digit, prints trailing sign for negative numbers.
^^^^	Prints in scientific notation (exponential) format.
$$	Prints leading $.
**	Fills leading spaces with *.
**$	Fills leading spaces with * and prints leading $.

String format characters:

&	Prints entire string.
!	Prints only the first character of a string.
\ \	Prints first n characters of a string where n is the number of spaces between the slashes + 2.

Other format characters:

_	Prints the following character as a literal.

Examples

```
USING$("TOTAL: #####",1000)          returns "TOTAL:  1000"
USING$("#####,.##",1000.21)          returns " 1,000.21"
USING$("####-",-123)                 returns " 123-"
USING$("$$###.##",121.95)            returns " $121.95"
USING$("**###.##",121.95)            returns "**121.95"
USING$("**$##.##",121.95)            returns "*$121.95"
USING$("**$####,.##-",-1021.95)      returns "**$1,021.95-"
USING$("&","TEST STRING")            returns "TEST STRING"
USING$("\  \","TEST STRING")         returns "TEST"
USING$("!","TEST STRING")            returns "T"
USING$("_###",5)                     returns "# 5"
```

See Also

PRINT USING, **DOLLAR$**

8.241 VAL Function

Converts a string to a number.

VAL(*string***)**

Arguments

string
String to convert.

Examples

```
VAL("100")      returns 100
VAL("-0.75")    returns -0.75
```

See Also

DEC, **STR$**

8.242 VARPTR Function (Version 1.4 or later)

Returns the address of a variable.

VARPTR(*variable***)**

Arguments

variable
The variable whose address is to be returned.

Remarks

The **VARPTR** function returns a number that identifies the location in memory of the specified variable.

Examples

```
VARPTR(CNT)     returns the address of the numeric variable CNT
VARPTR(NM$)     returns the address of the string variable NM$
```

See Also

VARPTR$

8.243 VARPTR$ Function (Version 1.4 or later)

Returns the address of a variable.

VARPTR$(*variable***)**

Arguments

variable
The variable whose address is to be returned.

Remarks

The **VARPTR$** function returns a string that identifies the location in memory of the specified variable.

Examples

```
VARPTR$(CNT)    returns the address of the numeric variable CNT
VARPTR$(NM$)    returns the address of the string variable NM$
```

See Also

VARPTR

8.244 VER$ Function

Returns the version number information.

VER$

8.245 VERIFY Function

Verifies a file name.

VERIFY(*filename***)**

Arguments

filename
File name to verify.

Remarks

The **VERIFY** function only checks that a file name is of the proper format. It does not check that the volume or file exists or that the file can be opened.

Examples

```
VERIFY("DATA:CONFIG.DAT")          returns -1 (true)
VERIFY("TEST")                     returns -1 (true)
VERIFY("*.TXT")                    returns 0 (false)
```

8.246 VOLINFO$ Function (standard version only)

Returns volume information.

VOLINFO$(*filespec***)**

Arguments

volume
Volume to return information for.

Remarks

The **VOLINFO$** function returns the volume information for the specified volume. The return string is the same format used by the default format of the **VOLUMES** statement. The information includes the name of the volume (first 15 characters), a space, default indicator ("*" if default, space if not), read-only indicator ("R" if read-only, space if not) and mapped indicator ("!" if NOT mapped, space if mapped).

If *volume* does not exist an empty string is returned.

Examples

```
VOLINFO$("HOME")                   returns "HOME           "
```

See Also

VOLUMES, **FILEINFO$**, **VOLUME$**

8.247 VOLUME$ Function

Returns the default volume or enumerates mapped volumes.

VOLUME$[**(***control***)**]

Arguments

control
Determines what volume will be returned.

Remarks

If *control* is omitted or -1 the default volume is returned. If *control* is 0 volume enumeration is initialized and the first volume is returned. Specify 1 for *control* to return the next volume. An empty string is returned when there are no more volumes. The volumes are returned in no particular order. If *control* is 2, a string containing all volumes separated by a comma is returned.

Examples

`VOLUME$(0)`	returns the first volume.
`VOLUME$(1)`	returns the next volume or an empty string if no more volumes.

See Also

VOLUMES

8.248 VOLUMES Function

Returns the number of mapped volumes.

VOLUMES

See Also

VOLUME$

8.249 WAITKEY$ Function (standard version only)

Returns a string corresponding to a key press.

WAITKEY$

Remarks

The **WAITKEY$** function waits for a key to be pressed.

See Also

INKEY$**, **INYN$

8.250 WEEKDAY Function

Returns the weekday from a time value.

WEEKDAY(*time***)**

Arguments

time
Time value whose weekday will be returned.

Remarks

The **WEEKDAY** function returns the day of the week as a number based on the following:

0 - Sunday
1 - Monday
2 - Tuesday
3 - Wednesday
4 - Thursday
5 - Friday
6 - Saturday

Examples

`WEEKDAY(MKTIME(2002,7,24))` returns 3

See Also

DAY, **HOUR**, **MINUTE**, **MONTH**, **SECOND**, **YEAR**

8.251 YEAR Function

Returns the year from a time value.

YEAR(*time***)**

Arguments

time
Time value whose year will be returned.

Examples

`YEAR(MKTIME(2002,7,24))` returns 2002

See Also

DAY, **HOUR**, **MINUTE**, **MONTH**, **SECOND**, **WEEKDAY**

9
Operators

9.1 Algebraic Operators

9.1.1 * Operator

Multiplies two numbers.

number1 ***** *number2*

Arguments

number1,*number2*
Numbers to be multiplied.

Examples

```
2 * 2
```
returns 4

9.1.2 + Operator

Adds two numbers.

number1 **+** *number2*

Arguments

number1,*number2*
Numbers to be added.

Examples

```
1 + 2
```
returns 3

9.1.3 - Operator

Subtracts two numbers or negates a number.

number1 **-** *number2*

- or -

-*number*

Arguments

number1,*number2*
Numbers to be added.

number
Number to be negated.

Examples

```
4 - 2          returns 2
- 2            returns -2
```

9.1.4 / Operator

Divides two numbers.

number1 **/** *number2*

Arguments

number1,*number2*
Numbers to be divided.

Examples

```
8 / 2          returns 4
```

9.1.5 \ Operator

Divides two numbers.

number1 **** *number2*

Arguments

number1,*number2*
Numbers to be divided.

Remarks

The **** operator rounds the two numbers to integers, divides them and returns the integer result.

Examples

```
9.8 \ 2.1          returns 5
```

9.1.6 ^ Operator

Raises a number to the power of an exponent.

number **^** *exponent*

Arguments

number
Numbers to be raised to a power.

exponent
Exponent used as power.

Examples

```
3 ^ 2          returns 9
```

9.1.7 DIV Operator

Divides one number by another number and returns the integer result.

number1 **DIV** *number2*

Arguments

number1,*number2*
Numbers to be divided.

Remarks

The **DIV** operator divides the two numbers and returns only the whole (integer) part of the quotient.

Examples

```
9.8 DIV 2.1          returns 4
```

9.1.8 MOD Operator

Divides one number by another number and returns the remainder.

number1 **MOD** *number2*

Arguments

number1,*number2*
Numbers to be divided.

Remarks

The **MOD** operator rounds the numbers to integers before dividing.

Examples

```
9.8 MOD 2.1          returns 0
```

9.2 Comparative Operators

9.2.1 < Operator

Compares two numbers or strings to determine if the first is less than the second.

number1 **<** *number2*

- or -

string1 **<** *string2*

Arguments

number1,*number2*
Numbers to be compared.

string1, *string2*
Strings to be compared.

Remarks

The **<** operator returns TRUE (-1) if the first argument is less than the second or FALSE (0) otherwise. String comparisons use ANSI collating.

Examples

```
2 < 3          returns -1 (true)
4 < 1          returns 0 (false)
5 < 5          returns 0 (false)

"A" < "B"      returns -1 (true)
"D" < "C"      returns 0 (false)
"E" < "E"      returns 0 (false)
```

9.2.2 <= Operator

Compares two numbers or strings to determine if the first is less than or equal to the second.

number1 **<=** *number2*

- or -

string1 **<=** *string2*

Arguments

number1,*number2*
Numbers to be compared.

string1, *string2*
Strings to be compared.

Remarks

The **<=** operator returns TRUE (-1) if the first argument is less than or equal to the second or FALSE (0) otherwise. String comparisons use ANSI collating.

Examples

```
2 <= 3          returns -1 (true)
4 <= 1          returns 0 (false)
5 <= 5          returns -1 (true)

"A" <= "B"          returns -1 (true)
"D" <= "C"          returns 0 (false)
"E" <= "E"          returns -1 (true)
```

9.2.3 <> Operator

Compares two numbers or strings to determine if the first is not equal to the second.

number1 **<>** *number2*

- or -

string1 **<>** *string2*

Arguments

number1,*number2*
Numbers to be compared.

string1, *string2*
Strings to be compared.

Remarks

The **<>** operator returns TRUE (-1) if the first argument is not equal to the second or FALSE (0) otherwise. String comparisons use ANSI collating.

Examples

```
2 <> 3          returns -1 (true)
4 <> 1          returns -1 (true)
5 <> 5          returns 0 (false)

"A" <> "B"      returns -1 (true)
"D" <> "C"      returns -1 (true)
"E" <> "E"      returns 0 (false)
```

9.2.4 = Operator

Compares two numbers or strings to determine if the first is equal to the second.

number1 **=** *number2*

- or -

string1 **=** *string2*

Arguments

number1,*number2*
Numbers to be compared.

string1, *string2*
Strings to be compared.

Remarks

The **=** operator returns TRUE (-1) if the first argument is equal to the second or FALSE (0) otherwise. String comparisons use ANSI collating.

Examples

```
2 = 3           returns 0 (false)
4 = 1           returns 0 (false)
5 = 5           returns -1 (true)

"A" = "B"       returns 0 (false)
"D" = "C"       returns 0 (false)
"E" = "E"       returns -1 (true)
```

9.2.5 > Operator

Compares two numbers or strings to determine if the first is greater than the second.

number1 **>** *number2*

- or -

string1 **>** *string2*

Arguments

number1,*number2*
Numbers to be compared.

string1, *string2*
Strings to be compared.

Remarks

The **>** operator returns TRUE (-1) if the first argument is greater than the second or FALSE (0) otherwise. String comparisons use ANSI collating.

Examples

```
2 > 3          returns 0 (false)
4 > 1          returns -1 (true)
5 > 5          returns 0 (false)

"A" > "B"      returns 0 (false)
"D" > "C"      returns -1 (true)
"E" > "E"      returns 0 (false)
```

9.2.6 >= Operator

Compares two numbers or strings to determine if the first is greater than or equal to the second.

number1 **>=** *number2*

- or -

string1 **>=** *string2*

Arguments

number1,*number2*
Numbers to be compared.

string1, *string2*
Strings to be compared.

Remarks

The **>=** operator returns TRUE (-1) if the first argument is greater than or equal to the second or FALSE (0) otherwise. String comparisons use ANSI collating.

Examples

```
2 >= 3          returns 0 (false)
4 >= 1          returns -1 (true)
5 >= 5          returns -1 (true)

"A" >= "B"      returns 0 (false)
"D" >= "C"      returns -1 (true)
"E" >= "E"      returns -1 (true)
```

9.3 Concatenation Operators

9.3.1 & Operator

Concatenates two strings.

string1 **&** *string2*

Arguments

string1, *string2*
Strings to be concatenated.

Examples

```
"ABC" & "DEF"          returns "ABCDEF"
```

9.3.2 + Operator

Concatenates two strings.

string1 **+** *string2*

Arguments

string1, *string2*
Strings to be concatenated.

Examples

```
"ABC" + "DEF"          returns "ABCDEF"
```

9.4 Logical Operators

9.4.1 AND Operator

Performs a conjunction operation.

number1 **AND** *number2*

Arguments

number1,*number2*
Numbers to be tested.

Remarks

The **AND** operator rounds the numbers to integers before performing the operation. Logical operators perform bit-wise operations, Boolean operations or tests on

multiple relations for making a decision. If the expressions evaluate to 0 or -1 the result is 0 or -1. Using values other than 0 for false and -1 for true may produce unexpected results.

The **AND** operator returns a result based on the following truth table:

number1	*number2*	**AND**
T	T	T
T	F	F
F	T	F
F	F	F

9.4.2 EQV Operator

Performs an equivalence operation.

number1 **EQV** *number2*

Arguments

number1,*number2*
Numbers to be tested.

Remarks

The **EQV** operator rounds the numbers to integers before performing the operation. Logical operators perform bit-wise operations, Boolean operations or tests on multiple relations for making a decision. If the expressions evaluate to 0 or -1 the result is 0 or -1. Using values other than 0 for false and -1 for true may produce unexpected results.

The **EQV** operator returns a result based on the following truth table:

number1	*number2*	**EQV**
T	T	T
T	F	F
F	T	F
F	F	T

9.4.3 IMP Operator

Performs an implication operation.

number1 **IMP** *number2*

Arguments

number1,*number2*
Numbers to be tested.

Remarks

The **IMP** operator rounds the numbers to integers before performing the operation. Logical operators perform bit-wise operations, Boolean operations or tests on multiple relations for making a decision. If the expressions evaluate to 0 or -1 the result is 0 or -1. Using values other than 0 for false and -1 for true may produce unexpected results.

The **IMP** operator returns a result based on the following truth table:

number1	*number2*	**IMP**
T	T	T
T	F	F
F	T	T
F	F	T

9.4.4 NOT Operator

Performs a bit-wise complement (1's compliment) operation.

NOT *number*

Argument

number
Number to be complimented.

Remarks

The **NOT** operator rounds the number to an integer before performing the operation. Logical operators perform bit-wise operations, Boolean operations or tests on multiple relations for making a decision. If the expression evaluates to 0 or -1 the result is 0 or -1. Using values other than 0 for false and -1 for true may produce unexpected results.

The **NOT** operator returns a result based on the following truth table:

number	**NOT**
T	F
F	T

9.4.5 OR Operator

Performs a disjunction (inclusive "or") operation.

number1 **OR** *number2*

Arguments

number1,*number2*
Numbers to be tested.

Remarks

The **OR** operator rounds the numbers to integers before performing the operation. Logical operators perform bit-wise operations, Boolean operations or tests on multiple relations for making a decision. If the expressions evaluate to 0 or -1 the result is 0 or -1. Using values other than 0 for false and -1 for true may produce unexpected results.

The **OR** operator returns a result based on the following truth table:

number1	*number2*	**OR**
T	T	T

T	F	T
F	T	T
F	F	F

9.4.6 XOR Operator

Performs an exclusive "or" operation.

number1 **XOR** *number2*

Arguments

number1,*number2*
Numbers to be tested.

Remarks

The **XOR** operator rounds the numbers to integers before performing the operation. Logical operators perform bit-wise operations, Boolean operations or tests on multiple relations for making a decision. If the expressions evaluate to 0 or -1 the result is 0 or -1. Using values other than 0 for false and -1 for true may produce unexpected results.

The **XOR** operator returns a result based on the following truth table:

number1	*Number2*	**XOR**
T	T	F
T	F	T
F	T	T
F	F	F

9.5 Precedence

Operators are shown from highest to lowest precedence.

^	Exponentiation
-	Negation
*, /	Multiplication, Division
\	Integer Division
DIV, MOD	Integer Quotient, Modulus
+, -	Addition, Subtraction
+ or &	String Concatenation
=, <>, <, >, <=, >=	Comparison
NOT	Logical NOT
AND	Logical AND
OR, XOR	Logical OR, Exclusive OR
EQV, IMP	Equivalence, Implication

10

Control and Editing Keys

10.1 Control Keys

ENTER	In Command (CMD) mode, executes the command, otherwise ends input and returns the buffer
INSERT	Toggles insert/replace mode
BREAK	In Command (CMD) mode, ignores characters in buffer, in Edit (EDT) mode stops editing and ignores any changes made to the buffer, otherwise stops execution of a statement or program. The BREAK key is CTRL+PAUSE/BREAK.
ESC	In Edit (EDT) mode, stops editing and ignores any changes made to the buffer, otherwise deletes all characters in the buffer
PAUSE	Pauses execution of a statement or program
F1	Displays NBASIC help or context help for controls in dialogs
F5	Runs the current program (Command (CMD) mode only)
F6	Steps to the next statement (Standard version only)
SHIFT+F6	Steps to the next line (Standard version only)
F8	Edits the immediate or program statement that caused the last error. (Version 1.4 or later)
F9	Recalls the previous command (Command (CMD) mode only). Pressing F9 in succession will recall previous commands from the command history buffer (Scrolls backward)
SHIFT+F9	Recalls the next command (Command (CMD) mode only). Pressing SHIFT+F9 in succession will recall the command next in the command history buffer (Scrolls forward)
F10	Displays the NBASIC system menu
CTRL+SHIFT+F10	Centers the NBASIC window
F12	Clears the screen (Command (CMD) mode only)
CAPS LOCK	Toggles caps lock
NUM LOCK	Toggles num lock
SCROLL LOCK	Toggles scroll lock

10.2 Editing Keys

LEFT	Moves cursor left one character
CTRL+LEFT	In Command (CMD) or Edit (EDT) modes, moves cursor to beginning of statement, otherwise moves cursor to beginning of buffer
RIGHT	Moves cursor right one character
CTRL+RIGHT	In Command (CMD) or Edit (EDT) modes, moves cursor to beginning of next statement, otherwise moves cursor to end of buffer
DELETE	Deletes character under cursor
CTRL+DELETE	Deletes all characters in buffer

HOME	Moves cursor to beginning of buffer
CTRL+HOME	Deletes characters from cursor to beginning of buffer
END	Moves cursor to end of buffer
CTRL+END	Deletes characters from cursor to end of buffer
UP	Moves cursor up one line or to beginning of buffer if only one line
DOWN	Moves cursor down one line or to end of buffer if only one line
BACKSPACE	Deletes character to left of cursor
TAB	Inserts spaces
SHIFT+TAB	Removes spaces

11
Error Codes

Error	Description
1	Access denied
2	Already dimensioned
3	Already open
4	Bad record number
5	Bad subscript
6	Command error
7	Const redefined
8	Cannot continue
9	/0 error
10	Duplicate label
11	ELSE without matching IF
12	END IF without matching IF
13	FIELD overflow
14	Bad file
15	File mode error
16	File not found
17	File not open
18	Function call error
19	Illegal constant definition
20	Input past end of file
21	Internal error
22	Invalid file name
23	Invalid volume
24	I/O error
25	Line too long
26	Log not open
27	Volume's mapped folder no longer exists
28	Maximum file size exceeded
29	Merge error
30	Mode error
31	NEXT without matching FOR
32	No printer connected
33	Not supported
34	Out of data
35	Out of memory
36	Overflow error
37	Renumber error
38	Resource not found
39	RESUME without ON
40	RETURN without GOSUB
41	Unable to start NBASIC
42	String too long
43	Syntax error
44	System error
45	Unable to create timer
46	Timer not set
47	Type mismatch
48	Undefined constant
49	Undefined function

50	Undefined label
51	Undefined line
52	Underflow error
53	Unexpected end of file
54	Volume full
55	Volume does not exist
56	Unable to open clipboard
57	Printer already open
58	Printer error
59	Printer not open

Appendix A
Hints and Tips

BASIC

Add a ; (semi-colon) after the arguments in a **PRINT** statement if you do not have to move to the next line, this is faster because no newline is performed.

To input a single character and wait for the ENTER key to be pressed, use `INPUT$(1)`.

To setup a one shot timer, place `TIMER STOP` or `ON TIMER GOTO 0` in the timer handler.

To stop a program that has BREAK set to off, Right click on the NBASIC screen and select Reset from the context menu.

To check for commands that cannot be executed in the shareware version of NBASIC, use the **CHKSW** statement.

To keep track of field widths for strings in **PRINT USING** statements, use string formats like "/2345/" (the beginning and ending /'s count).

NBASIC can load ASCII programs with no line numbers; lines are added to the program in order that they appear in the file.

Before adding line numbers to a program that does not contain line numbers, renumber the program.

Programming

To execute shareware or standard version specific commands, check the value at PEEK(23) to determine version (e.g. `IF PEEK(23)=0 THEN DEC A ELSE A=A-1`).

To execute version specific commands, check the value at PEEK(36) to determine version. PEEK(36) returns the release number of NBASIC (1-Version 1.0, 2-Version 1.0.1, 3-Version 1.1, 4-Version 1.2, 5-Version 1.2.1, 6-Version 1.3, 7-Version 1.4, 8-Version 1.5, 9-Version 1.5.1, 10-Version 1.5.2). New commands in the online help and reference guides include the minimum version information.

Appendix B

System Requirements

NBASIC requires Windows 95, Windows 98, Windows ME, Windows NT 4.0 or later, Windows 2000, Windows XP or Windows Vista. It will not run on Windows 3.x or Windows NT 3.1 or 3.5.

Version 4.71 or later of the Windows Common Control library is required. This Library is part of Windows 98, Microsoft Internet Explorer 4.0, and is available as a separate download from the Microsoft web site. The library is not required for Windows 2000 or later.

Appendix C

Installing/Uninstalling NBASIC

Installing NBASIC

To install NBASIC from the internet/e-mail distribution package, find the file from the location where it was saved, and double click it.

To install NBASIC from the distribution disks, insert the Setup disk and choose Run from the Start menu, type a:setup, and press the ENTER key. Follow the directions in the Setup program.

It is strongly recommended that you accept the installation default settings.

Uninstalling NBASIC

Activate the Control Panel, double click Add/Remove Programs, and double click on the NBASIC list box entry.

Appendix D

Support

For support, send e-mail to sylvaware@mindspring.com.

When reporting a problem, please include the following information:

1. Is the problem reproducible? If so, how? If possible, send a program to demonstrate the problem.
2. What version of Windows are you running (Windows 95, Window NT 4.0, Windows 2000, etc)?
3. What version of NBASIC are you running (to see what version of NBASIC you have, choose About... from the context menu by right-clicking with the mouse or type ABOUT at the command prompt)?
4. If a dialog box with an error message was displayed, please include the full text of the dialog box, including the text in the title bar.

You can press F1 at any time while NBASIC is active to get help or type HELP at the command prompt.

Visit us online for more information on NBASIC: http://sylvaware.home.mindspring.com.

Index

G

H

I

K

L

Notes

Notes