

# ***REFERENCE***

This section provides a reference for the functions and statements of *WinBasic*, organized alphabetically.

## ***Overview***

### ***Data definition***

DIM	Define array
LOCAL	Declare variable local
STATIC	Declare variable static
RECTYPE	Define recordtype
RECVAR	Define record item
ENDREC	End recordtype definition
RECORD	Define record

### ***Program control***

IF	If statement
FOR	For loop
WHILE	While loop
REPEAT	Repeat loop
BREAK	Exit loop
STOP	End programm
GOTO	Goto label
SUBROUTINE	Start of subroutine
ENDSUB	End of subroutine
DECLARE	Declare subroutine
RETURN	Return from subroutine
GOSUB	Call subroutine

## ***Data IO***

OPEN	Open file
CLOSE	Close file
INPUT	Read data
LINE INPUT	Read data
PRINT	Print data (file, screen)
LPRINT	Print data (printer)
WRITE	Write data
GET	Read random file
PUT	Write random file
CHDIR	Change directory
GETDIR	Get directory
ERASE	Delete file
RENAME	Rename file
INKEY\$	Read keyboard

MOUSE	Get mouse data
-------	----------------

## ***Mathematical functions***

SIN	Sine
COS	Cosine
TAN	Tangent
ATN	Arctangent
EXP	Exponential function
LOG	Natural logarithm
SQR	Square root

## ***String functions***

SET\$	Set characters in string
DELETE\$	Delete characters in string
INSERT\$	Insert characters in string
INSTR%	Search partial string
LEFT\$	Form left partial string
RIGHT\$	Form right partial string
MID\$	Form partial string
TRIM\$	Cut trailing blanks
LTRIM\$	Cut heading blanks
LCASE\$	Convert to lowercase
UCASE\$	Convert to uppercase
LEN%	Count string length
ASC%	Convert character to ASCII value
CHR\$	Convert value to character
STR\$	Convert number to string

VAL	Convert string to real
VAL%	Convert String to integer
DOSTOWIN\$	Conversion IBM to ANSI
WINTODOS\$	Conversion ANSI to IBM

## ***Graphics***

CLS	Clear screen
COLOUR	Define colours
CURSOR	Define cursor
LOCATE	Set cursor in character units
STYLE	Line and brush style
MAPMODE	Set mapping mode
MOVETO	Set cursor in graphic units
LINE TO	Draw line
ARC	Draw arc or circle
ELLIPSE	Draw ellipse
PIE	Draw pie
RECT	Draw rectangle
BITMAP	Draw bitmap
FONT	Choose font

## ***Miscellaneous***

REM	Comment
'	Comment
ERROR%	Get runtime error
EXEC	Start program
INTERRUPT	Generate interrupt

TIMEDATES\$	Get time and date
TRACE	Trace program

## ***Windows functions***

FILES	Choose file
DIALOG	Create dialog box
DLGITEM	Control dialog item
FORM	Data entry form for records
MENU	Set Menu
MENUITEM	Modify menu item
MESSAGEBOX	Display messagebox
SHOW	Show program window
SIZE	Modify window size
CLIP	Read/write clipboard



## ***Runtime errors***

When a program is run, several errors can occur. *WinBasic* stores the code number of the last occurred error; this number can be obtained with the built-in function `ERROR%( )`.

### ***Error codes***

- |   |   |
|---|---|
| 1 | A file is already open with this identifier |
| 2 | File not open                               |
| 3 | Read or write after end of file             |
| 4 | Too many files open                         |
| 5 | Out of memory (random access file)          |



## ***Reference***

### ***Graphics device number***

Some graphic statements may have a device number which identifies the output device. The following numbers are available:

- |   |                    |
|---|--------------------|
| 1 | Screen             |
| 2 | Printer            |
| 3 | Screen and printer |

If the device number is omitted, output is to the screen.

Output to the printer is buffered and only appears after a form feed character (see LPRINT).

# ARC

## Usage:

Arc *r%*, *beg%*, *end%* [, *G%*]

*r%* Integer

*beg%*: Integer

*end%*: Integer

*G%* Graphics device number

## Description:

ARC draws an arc or a circle. *r%* is the radius of the arc, *beg%* and *end%* are the start and end angles in degrees. The angles are measured against a downward pointing y-axis. The mid point of the arc lies on the current coordinates (see MOVETO). If the start and end angle are equal or 0 and 360 degrees, a circle is drawn. The arc is drawn counterclockwise in the current line style, pen width and colour.

## Example:

```
MoveTo 300, 300
```

```
Arc 100, 90, 270      'Result: a half circle
```

## See also:

ELLIPSE, COLOUR, STYLE

# ASC%



**Usage:**

Asc%(Str\$)

Str\$: String

**Description:**

ASC%() is a function which returns the ASCII code of the first character in *Str\$*. This is the inverse function of CHR\$( ).

**Example:**

```
MoveTo 300, 300
```

```
Arc 100, 90, 270      'Result: a half circle
```

**See also:**

CHR\$( )

## ***ATN( )***

**Usage:**

Atn(*x*)

*x*: Numeric value

**Description:**

ATN( ) returns the arctangent in radians of the parameter *x*.

## ***BITMAP***

**Usage:**

Bitmap *File\$* [, *G%*]

File\$: String

G%: Graphics device number

**Description:**

Bitmaps are drawings created by the Windows Paint program, copied into the clipboard and saved to disk file using the BITMAP.EXE utility. The BITMAP statement draws such an object stored in the file *File\$* on the screen or printer. The upper left corner of the bitmap lies on the current coordinates (see MOVETO)

## ***BREAK***

**Usage:**

Break

**Description:**

BREAK leaves the currently executing loop unconditionally. The next statement executed is the first statement after the end of the loop.

BREAK is not allowed outside a loop.

**See also:**

FOR, WHILE, REPEAT

# CHR\$( )

## Usage:

Chr\$(*n%*)

*n%*: Integer

## Description:

The function CHR\$( ) returns the character which is represented by the code *n%* in the ANSI character set. This is the inverse function of ASC%( ).

## Example:

```
Print Chr$(65)           'Result is: A
```

## See also:

ASC%( )

# CLIP

## Usage:

Clip Get *file\$*

Clip Put *file\$*

*file\$*: String

## Description:

CLIP reads or writes to the Windows clipboard. CLIP GET copies the contents of the clipboard into the file *file\$*. CLIP PUT copies the file *file\$* into the clipboard.

The clipboard is a temporary storage location accessible to all applications. It is usually used to exchange data between different applications or within one application. Most Windows applications allow access to the clipboard through an **Edit** menu containing functions like **Copy**, **Cut** and **Paste**.

Different types of data can be stored in the clipboard. Currently *WinBasic* supports the formats CF\_TEXT, CF\_SYLK and CF\_DIF. All three formats are textual. CF\_TEXT is common text. CF\_SYLK and CF\_DIF are special text formats, namely "Microsoft Symbolic Link Format" and "Software Arts Data Interchange Format".

If the clipboard contains data in any other format, its contents is ignored. Data is always written in CF\_TEXT format.

**Example:**

```
Close #3
```

**See also:**

OPEN

## CLOSE

**Usage:**

Close *Ident*

Ident: See OPEN

**Description:**

The file that was opened with the identifier *ident* is closed. No further I/O operations are allowed on that file. *Ident* is free to be used to open other files.

**Example:**

Close #3

**See also:**

OPEN

## CLS

**Usage:**

Cls [*n%*]

**Description:**

CLS clears the program window. The background colour can be optionally set with *n%*. Possible values are:

- |   |            |
|---|------------|
| 0 | White      |
| 1 | Light grey |
| 2 | Grey       |
| 3 | Dark Grey  |
| 4 | Black      |

**Example:**

```
Cls 4      'Black
```

## COLOUR

**Usage:**

Colour *f%*[,*b%*]

*f%*     Integer

*b%*     Integer

**Description:**

COLOUR sets the current text and graphics output colour. *f%* is the foreground colour, *b%* the background colour. If the background colour is omitted, it remains unchanged. The possible colours are:

0	Black
1	Blue
2	Green
3	Cyan
4	Red
5	Magenta
6	Yellow/Brown
7	White

With graphics output (LINETO, ARC, ...) the background colour is used to fill the gaps (for hatches or dotted lines) while the

foreground colour is used for the drawn object itself. With text output, the text is written in the foreground colour on the background colour.

If the background colour  $b\%$  is negative, it is only used for graphics (with its absolute value); text output does not alter the existing background (transparent mode).

By default the foreground colour is black and the background colour is white.

**Example:**

```
Colour 7, 1      'White on Blue
```

## **COS()**

**Usage:**

$\text{Cos}(x)$

x:                Numeric value

**Description:**

The function COS( ) returns the cosine of the parameter  $x$ , which must be given in radians.

## **CURSOR**

**Usage:**

Cursor *n%*

*n%*: Integer

**Description:**

This statement sets the shape of the mouse cursor in your program's window. Possible values for *n%* are:

0	Arrow (default)
1	I-Beam
2	Hourglass
3	Cross

## **DECLARE**

**Usage:**

Declare *Name*[[*[[VAR]**Par*[,...]]]]

Name: Subroutine name

VAR: reserved word

Par: Variable name

**Description:**

Subroutines must be declared before they can be used in a program. DECLARE declares the name and parameters of a subroutine without having to define its content. The complete definition must follow later in a SUBROUTINE statement. The parameters given in the DECLARE and SUBROUTINE statements must correspondent; differences are not checked and



may result in undefined behaviour.

**Example:**

```
Declare Info$(s$)
Subroutine Thing()
...
Gosub Info$("Thing")
...
Endsub
...
Subroutine Info$(s$)
Print s$
Endsub
...
Gosub Thing()
```

**See also:**

SUBROUTINE

## ***DELETE\$***

**Usage:**

Delete\$ *Str\$, index%, number%*

Str\$: String variable  
index%: Integer  
number% Integer

**Description:**

DELETE\$ deletes *number%* characters in *Str\$* beginning at position *index%*. The first character of the string is at index

position 1. The length of *Str\$* is decreased.

**Example:**

```
A$ = "ABcd"  
Delete$ A$, 2, 2  
Print A$                                'Result is: Ad
```

## DIALOG

**Usage:**

1. Dialog *x%*, *y%*, *dx%*, *dy%*, *IDitem%*, *class%*, *text\$*
2. Dialog
3. Dialog @

<i>x%</i> , <i>y%</i> :	Integer
<i>dx%</i> , <i>dy%</i> :	Integer
<i>IDitem%</i> :	Integer
<i>class%</i> :	Integer
<i>text\$</i> :	String

**Description:**

DIALOG creates a dialog box or terminates an existing one. The first form of the statement defines the items contained in the dialog box. A statement of this form is given for every item in a dialog box; this list of statements is terminated by a statement of the second form, which also causes the subroutine to suspend and the dialog box to appear on the screen. The third form of the statement terminates processing of the dialog box and removes it from the screen.

Each dialog box has one dedicated subroutine that "drives" it. A list of DIALOG statements of the first form are placed at the start of the subroutine. This list is terminated by a DIALOG statement of the second form. This is followed by a sequence of Windows labels, which mark entry points for the code that defines the behaviour of the dialog box. These Windows labels correspond to the *IDitem%* of each dialog item; when the user activates an item in the dialog box, the message code *IDitem%* is sent to the subroutine.

The first DIALOG statement in the subroutine defines the border of the dialog box. For this, *class%* is set to 0. *x%* and *y%* are the coordinates of the upper left corner of the dialog box, *dx%* and *dy%* its horizontal and vertical size. In this case, *IDitem%* has no meaning and should be set to -1. The string *text\$* becomes the title of the dialog.

The subsequent DIALOG statements define the items in the dialog box. *x%* and *y%* are the coordinates of the upper left corner of the item relative to the upper left corner of the dialog box, *dx%* and *dy%* its horizontal and vertical size. *IDitem%* must be set to a unique number, which is sent as a message to the subroutine when the item is activated by the user. The value of *IDitem%* must be between 3 and 99. When the Enter key is pressed, Windows sends 1 as a message, when Esc is pressed a 2 is sent. Numbers higher than 99 are reserved for other uses.

Note that *x%*, *y%*, *dx%* and *dy%* are not measured in pixels; rather, they are related to the system font. On the x-axis, one unit corresponds to 1/4 the width of a character, on y-axis it is 1/8 of the height of a character.

*class%* is used to define the type of item displayed:

- 0 BORDER (first statement)
- 1 CHECKBOX: box with check mark
- 5 LTEXT: left aligned constant text
- 10 GROUPBOX: box around group of radio buttons
- 11 PUSHBUTTON
- 12 RADIOBUTTON
- 13 DEFPUSHBUTTON: default push button
- 18 LEDIT: left aligned edit field

Items of type LTEXT and GROUPBOX cannot be activated by the user and therefore cannot generate messages. Their *IDitem%* should be set to -1.

CHECKBOX consists of a small square box which can contain a check mark, together with title defined by *text\$*. Since a check box always has the same size, *dx%* and *dy%* define the available space for *text\$*. The message *IDitem%* is sent when the check box is activated (clicked).

LTEXT generates the left aligned textual constant *text\$*. The item cannot be activated by the user and therefore cannot send a message, set *IDitem%* to -1.

GROUPBOX is usually used as a border around a group of radio buttons to show the user that they form a group. The item cannot be activated by the user and therefore cannot send a message, set *IDitem%* to -1.

PUSHBUTTON generates a push button with title *text\$*. When the button is "pushed", the message *IDitem%* is sent to the dialog box subroutine.

RADIOBUTTON generates a radio button. A radio button is a small circle which can be filled with a black spot to indicate that it is "on". Radio buttons are generally used in groups in which only one button is on while the others are off. They serve to let the user choose one option from a set of alternatives. *text\$* is the title of the generated button. When activated, the button sends the message *IDitem%* to the dialog box subroutine.

DEFPUSHBUTTON differs from PUSHBUTTON in that is supposed to be the default response of the user. A DEFPUSHBUTTON has a thicker border than PUSHBUTTON. There should be only one DEFPUSHBUTTON in any one dialog box. The dialog box subroutine should be programmed in such a way that pressing the Enter key (which sends the message code 1) gives the same response as pressing the DEFPUSHBUTTON, which sends message *IDitem%*.

LEDIT generates a left aligned edit field. *text\$* is displayed in the edit field and can be edited by the user. To enter characters into the edit field it must have the input focus, which it gets by the user activating the field or by the DLGITEM statement. It is not a good idea to respond to the message *IDitem%* !

As long as the dialog box is active the main program is inactive. However, the user may continue to work with other programs while the dialog box is active.

**Example:**

See example program DIALOG.BAS

**See also:**

DLGITEM

## ***DIM***

**Usage:**

```
Dim Name(len[,len..])[,Name(..)..len]
```

Name: Array name

len: Integer constant

**Description:**

DIM defines the size of an array. *Name* is a valid variable name, *len* the length of the array. During the execution of a program the individual variables of an array can be accessed by an index value between 1 and *len*. Arrays may be multidimensional. The type of an array is defined by the type of *Name*. Any simple data type is allowed. A DIM statement may define more than one array; the individual definitions are separated by commas.

**Example:**

```
Dim Int%(10,10), Str$(20)  
Str$(20) = "Assignment to Item 20"
```

## ***DLGITEM***

**Usage:**

1. DlgItem *id%*, *text\$*
2. DlgItem *id%*, *id1%*, *id2%*, *mode%*, *text\$*

*id%*: Integer  
*id1%*, *id2%*: Integer  
*mode%*: Integer  
*text\$*: String variable

**Description:**

The first form of DLGITEM reads the current contents of an edit field in a dialog box. *id%* is the identifier of the edit field as defined by a DIALOG statement. *text\$* is a string variable which receives the contents of the field.

The second form of DLGITEM is used to modify an item in a dialog box. There are three distinct modes of use:

1. *id%* is the identifier of an edit field. *text\$* becomes the new contents of the edit field. *id1%* and *id2%* must be 0. If *mode%* is set to 2, the edit field also gets the input focus, which means it collects key strokes. Otherwise, *mode%* should be 0.
2. *id%* is the identifier of a radio button. *text\$* becomes the new title of the button. If *mode%* is set to 4, the radio button *id%* is set "on", all other radio buttons with identifiers between *id1%* and *id2%* are set "off".
3. *id%* is the identifier of a check box. *id1%* and *id2%* must be 0, *text\$* becomes the new title of the check box. If *mode%* is set to

4, a check mark is set in the box, otherwise it is cleared.

Note: If *text\$* is set to Chr\$(1) the text of the item remains unchanged.

**Example:**

```
DlgItem 6, a$      'Reads edit field 6
DlgItem 6, 0, 0, 2, "abc" 'Set edit field 6
                        'to "abc"

DlgItem 13, 10, 15, 4, Chr$(1)
                        'Set Radiob. 13

DlgItem 19, 0, 0, 4, Chr$(1)
                        'Set Checkbox 19
```

**See also:**

DIALOG

## ***DOSTOWIN\$( )***

**Usage:**

DosToWin\$(*Str\$*)

Str\$: String

**Description:**

Windows uses the ANSI character set for text; MS-DOS uses the "IBM Extended Character Set". These character sets are identical in many cases (A-Z, a-z, 0-9 for example) but have some differences, especially the international and symbol characters. The function DOSTOWIN\$( ) converts a string from the MS-



DOS character set to the Windows character set. This function is useful for working with text files that were created by a DOS program.

**Example:**

```
Open "I", #1, "Text.dat"
While error%() <> 3
  Line Input #1, a$
  a$ = DosToWin$(a$)
  Print a$
Wend
Close #1
```

**See also:**

WINTODOS\$()

## **ELLIPSE**

**Usage:**

Ellipse *dx%*, *dy%*, *start%*, *end%* [, *G%*]

*dx%*, *dy%*: Integer

*start%*, *end%*: Integer

*G%*: Graphics device number

**Description:**

ELLIPSE draws an ellipse or circle centered on the current graphics output coordinates. *dx%* and *dy%* define the size of the ellipse; if they are equal, then a circle is drawn. *start%* and *end%* define the start and end angles in degrees; if they are set to 0 and

360, then a complete ellipse is drawn. The ellipse is drawn counterclockwise in the current line style, pen width and screen colours.

**Example:**

```
MoveTo 300, 300  
Ellipse 100, 200, 0, 360
```

**See also:**

ARC

## ***END***

**Usage:**

End

**Description:**

END ends the execution of a program

**See also:**

STOP

## ***ENDREC***

**Usage:**

Endrec

**Description:**

Terminates the sequence of RECVAR statements used to define a record type

**See also:**

RECTYPE, RECVAR

## ***ERASE***

**Usage:**

Erase *Name\$*

Name\$:       String

**Description:**

ERASE erases the file with name *Name\$* from the disk. *Name\$* can be a fully qualified filename with drive and directory.

**Example:**

```
Erase "c:\mydir\abc.txt"
```

## ***ERROR%( )***

**Usage:**

Error%( )

**Description:**

The function ERROR%( ) returns the last generated runtime error. Call this function liberally around I/O statements.

**Example:**

```
If Error%() <> 0 Then
  Print "Runtime error"
Endif
```

## **EXEC**

**Usage:**

Exec *Program\$, Com\$, show%*

Program\$: String  
Com\$: String  
Show%: Integer

**Description:**

EXEC executes other Windows programs. *Program\$* is the name of the program to start, including a path name if necessary.

*Program\$* will only be started if it is not already running. *Com\$* is the command line which is passed to the program after loading. *show%* determines how the main window of the started program is shown at start (icon, overlapped, full). Refer to SHOW for a discussion of the possible values of the *show%* parameter.

**Example:**

```
Exec "Notepad.exe", "WIN.INI", 1
```

**See also:**

SHOW, STOP

## ***EXP()***

### **Usage:**

$\text{Exp}(x)$

x:                      Numeric value

### **Description:**

EXP( ) returns the exponential function of the parameter  $x$ , *i.e.* to the power of  $x$ . The parameter must be less than 89.

## ***FILES***

### **Usage:**

Files *Dir*\$, *Filespec*\$, *File*\$

Dir\$: String

Filespec\$: String

File\$: String

### **Description:**

FILES lists all the files in the directory *Dir*\$ that match the pattern *Filespec*\$ in a list box. The user can select a filename or enter a new one.

Afterwards, *File*\$ contains the name of the chosen file or is

empty, if the dialog was cancelled.

**Example:**

```
Files "\winbasic", "*.bas", file$  
If file$ <> "" Then  
...  
Endif
```

## **FONT**

**Usage:**

Font *height%*, *width%*, *weight%*, *style%*, *family%*

height%	Integer
width%	Integer
weight%	Integer
style%	Integer
family%	Integer

**Description:**

In Windows, text may be written in various fonts on the screen or printer. The FONT statement generates a font which is used for subsequent output of text.

A font family is selected with *family%*, which defines the type face of the text characters:

- |   |             |
|---|-------------|
| 1 | Times Roman |
| 2 | Helvetica   |
| 3 | Swiss       |

4	Script
5	Roman
6	System (default)
7	Courier

*height%* and *width%* define the size points of the characters (1 point is about 1/72 inch).

*weight%* defines the stroke of the font. There are currently two values available:

400	Normal
700	Bold

All other values are truncated to these.

Special properties of the generated font can be defined with the *style%* parameter:

0	Normal
1	Italic
2	Underlined
3	Italic and underlined

**Example:**

```
Font 18, 12, 700, 0, 1
Print "Hello Windows"
```

The font Times Roman bold is generated with height 18, width 12.

# **FOR ... NEXT**

# **FOR ... ENDFOR**

## **Usage:**

For Counter = *Beg* To *End* [Step *Step*]

...

Endfor

Counter: Integer or long variable

Beg, End, Step: Integer or long

## **Description:**

The FOR statement creates a program loop. The statements between FOR and ENDFOR are repeated under control of the variable *Counter*. At first *Counter* is set to *Beg*. After each loop its value is increased by *Step*, which is 1 by default. When *Counter* equals *End*, the loop is left at ENDFOR.

NEXT is a synonym for ENDFOR.

*Beg*, *End* and *Step* may be variables or constants. They should all be of the same data type to avoid type conversions.

*Counter*, *Beg*, *End* and *Step* should be integers or longs. Reals and doubles are allowed. However, the accuracy of real and double expressions is not absolute, so a comparison of *Counter* and *End* cannot be guaranteed to terminate the FOR loop correctly.



FOR loops can be terminated independently of *Counter* by using the BREAK statement. If a BREAK statement is executed while the program is executing a FOOR loop, an immediate jump to the statement following ENDFOR is made.

**Example:**

```
For i% = 1 To 20
  s% = s% + i%
  If s% > 100 Then Break      ' leave loop
  Print "i%=";i%*10
Endfor
```

**See also:**

BREAK

## FORM

**Usage:**

Form *Rec*, *Prompt\$*, *Answer%* [, *x%*, *y%*]

Rec:	Record
Format\$:	String
Answer%:	Integer
x%, y%:	Integers

**Description:**

FORM builds a data entry form for Record *Rec*. *Prompt\$* contains the title of the form and the titles of individual fields, separated by semicolons. *x%* and *y%* define the upper left corner

of the data entry form; if omitted, default values are used.

After returning, *Answer%* holds the identifier of the button the user pressed to leave the entry form:

- 1      OK Button or Enter key
- 2      Cancel button or Esc key

If a field of *Rec* should not be edited by the user, its title in *Prompt\$* can be enclosed in parentheses.

**Example:**

```
Rectype Partrec
Recvar Partnr$ 16
Recvar Partname$ 30
Recvar Price 8:2
Endrec

Record Partrec part
s$ = "Part; (Partno:);Part name;;Price:;"
Form part, s$, Button%
```

## **GET**

**Usage:**

Get *Ident*, *Recnr%*, *Rec*

Ident: See OPEN

Recnr%: Integer

Rec: Record

**Description:**

GET reads data from a random access file. The file must have been opened by OPEN as a random access file. Random access files consist of a contiguous sequence of records of fixed length. The records are read and written as whole units. Records are identified by a record number. The record with record number *Recnr%* read by GET is stored into the record *Rec*.

Runtime errors 2, 3 and 5 may occur.

**Example:**

```
Get #3, 101, Rec
```

**See also:**

PUT, OPEN

## GOSUB

**Usage:**

```
Gosub Name[(Par)[,...]]
```

Name: Subroutine name

Par: Variable or constant

**Description:**

GOSUB calls the subroutine *Name*. The execution of the program continues at the first executable statement of the subroutine. After leaving the subroutine the program execution continues with the next statement following GOSUB.

If the called subroutine *Name* has no parameters, then the name can optionally be followed by a pair of parentheses.

If the subroutine has parameters, the actual parameters are listed between the parentheses, separated by commas. Their type and number must correspond with the formal parameters in the SUBROUTINE statement for *Name*.

**Example:**

see SUBROUTINE

**See also:**

SUBROUTINE, RETURN

## GOTO

**Usage:**

Goto *Label*

**Description:**

GOTO makes an unconditional jump to the statement which follows the label *Label*. *Label* is any alphanumeric string with a trailing colon. No statements are allowed on a line after a label.

**Example:**

```
If a% = 0 Then Goto TheEnd
...
TheEnd:
Stop
```

**See also:**

## ***IF***

### **Usage:**

1. If *LogExp* Then *Statements*
2. If *LogExp* Then  
    *Statements*  
    [Elseif *LogExp* Then  
        *Statements*]  
    [Else  
        *Statements*]  
    Endif

*LogExp*:                Logical expression  
*Statements*:    One or more statements

### **Description:**

The IF statement has two forms. In the first form the whole statement is on one line of program text. If the logical expression *LogExp* is true, the *Statements* are executed; if *LogExp* is false, the statements are not executed.

*Statements* may be several statements separated by colons, executed only if *LogExp* is true.

In the second form there are no statements after THEN. Instead,

one or more statements follow on several lines of the program text until the IF clause is terminated by an ENDIF statement, or an ELSE or ELSIF statement appears.

The IF construct can have several ELSEIF branches and one ELSE branch. No further ELSEIF statements may follow an ELSE statement. The statements between ELSE and ENDIF are executed if the *LogExp* of the IF statement and all ELSEIF statements (if any) are false.

In practice, *LogExp* can be any expression, not just a logical one.

**Example:**

```
Input "a%=", 1, a%
If a% = 1 Then
  Gosub Sub1()
Elseif a% = 2 Then
  Gosub Sub2()
Elseif a% = 3 Then
  Gosub Sub3()
Else
  Print "None of Them"
Endif
```

## ***INKEY\$( )***

**Usage:**

*a\$* = Inkey\$( )

*a\$*:     String variable

**Description:**

INKEY\$( ) checks whether or not a character has been typed at the keyboard since the last execution of INKEY\$( ). If true, this character is returned in *a\$*, otherwise an empty string "" is returned. INKEY\$( ) does not wait for input like INPUT but returns immediately.

*a\$* only contains printable characters or one of the function keys F1 to F9 and Shift F1 to Shift F9. Other keys are ignored. The simple function keys are returned as codes 187 - 195, the shifted function keys as codes 210 - 220.

**Example:**

```
a$ = Inkey$( )  
If Asc%(a$) = 195 Then Break 'F9 = End
```

## **INPUT**

**Usage:**

1. Input *Prompt\$, Len%, Var*
2. Input *Ident, Var [,Var...]*

Prompt\$: String  
Len%: Integer  
Var: Variable Types  
Ident: see OPEN

**Description:**

INPUT reads data from a small entry form (first form) or from a

file (second form).

The first form of INPUT is an alternative to a dialog box, requiring less programming. *Prompt\$* is issued as a prompt for data entry by the user. The prompt is followed by an edit field of length *Len%* containing the current value of the variable *Var*, which can be edited by the user.

The second form of INPUT statements reads one or more variables from the text file opened with identifier *Ident*. Commas and linefeeds are used to delimit the individual data items. Data items may be surrounded by a pair of double quotes. This allows string data to contain commas, which otherwise would be treated as separators.

Runtime error 2 may occur.

**Example:**

```
Input "Please enter Width:", 6, width%  
...  
Input #1, a%, b, c$
```

**See also:**

OPEN, LINE INPUT

## ***INSERT\$( )***

**Usage:**

Insert\$ *String\$, Index%, Partstring\$*



String\$: String variable  
Index%: Integer  
Partstring\$: String

**Description:**

INSERT\$ inserts *Partstring\$* at position *Index%* in *String\$*. The length of *String\$* is increased by the length of *Partstring\$*.

**Example:**

```
A$ = "Ad" : Part$ = "Bc"  
Insert$ A$, 2, Part$  
Print A$ 'Result is: ABcd
```

**See also:**

SET\$

## INSTR%( )

**Usage:**

Instr%(*String\$, Partstring\$*)

String\$: String  
Partstring\$: String

**Description:**

INSTR%( ) checks whether *Partstring\$* is included in *String\$*. If true, the position of the first character of *Partstring\$* within *String\$* is returned, otherwise 0 is returned.

**Example:**

```
A$ = "ABcd"  
index% = Instr%(A$, "Bc")  
Print index%           'Result is: 2
```

## ***LCASE\$( )***

### **Usage:**

Lcase\$(*String\$*)

String\$:       String

### **Description:**

LCASE\$ returns a string containing *String\$* converted to lower case characters. *String\$* remains unchanged.

### **Example:**

```
B$ = "ABcd"  
Print B$           'Result is: ABcd  
B$ = Lcase$(B$)  
Print B$           'Result is: abcd
```

### **See also:**

UCASE\$( )

## ***LEFT\$( )***

### **Usage:**

Left\$(*String\$,Number%*)

String\$:               String  
Number%:     Integer

**Description:**

LEFT\$( ) returns a partial string consisting of the leftmost *Number%* characters from *String\$*. *String\$* remains unchanged. If *String\$* has less than *Number%* characters it is simply copied.

**Example:**

```
A$ = Left$("ABcd",2)
Print A$                       'Result is: AB
```

**See also:**

RIGHT\$( ), MID\$( )

## LEN%( )

**Usage:**

Len%(*String\$*)

String\$:       String

**Description:**

LEN%( ) returns the number of characters in *String\$*.

**Example:**

```
Print Len$("ABcd")               'Result is: 4
```

# ***LINE INPUT***

## **Usage:**

Line Input *Ident*, *Var* [*Var...*]

Ident: see OPEN

Var: Variable

## **Description:**

LINE INPUT works like the second form of INPUT but expects only linefeeds to be used of separators.

Runtime error 2 may occur.

## **See also:**

OPEN INPUT

# ***LINE TO***

## **Usage:**

LineTo *x%*,*y%* [*G%*]

*x%*, *y%*: Integers

*G%*: Graphics device number

## **Description:**

LINE TO draws a line from the current coordinates (see MOVETO) to the coordinates *x%*, *y%*. Thereafter, these

coordinates become the current coordinates, as if they were set by MOVETO.

The line is drawn with the current line style and pen width (see STYLE) and with the current colours.

**Example:**

```
MoveTo 50, 50
LineTo 100, 50
LineTo 50, 100
LineTo 50, 50
```

**See also:**

COLOUR, STYLE

## LOCAL

**Usage:**

Local *Name* [*Name*...]

Name: Variable Name

**Description:**

The variable *Name* is defined as local to a subroutine. The variable is unknown outside the subroutine. In contrast, variables which are used in a subroutine without a LOCAL definition are 'global': they can also be accessed by the main program and other subroutines. LOCAL variables coexist with global variables of the same name defined by other parts of the program; they occupy different places in memory.

LOCAL statements must be placed at the beginning of a subroutine. They may not have the name of a formal parameter of the subroutine. Several variables may be defined, separated by commas.

LOCAL variables are volatile; their value is not preserved between calls to the subroutine. Their value on entry to the subroutine is undefined.

**Example:**

```
Subroutine Sub1(x%)  
  Local r, a1%  
  Local s$  
  ...  
Endsub
```

**See also:**

STATIC

## LOCATE

**Usage:**

Locate *m%*, *n%*

*m%*, *n%*:                      Integers

**Description:**

LOCATE sets the current coordinates of the screen to column *m%* and row *n%* in terms of characters. The height and the width

of the current character font is used for the calculation.  
Subsequent text output begins at this position.

LOCATE also sets the starting point of the graphics functions.  
LOCATE and MOVETO/LINETO both set the current  
coordinates, LOCATE in units of characters, the latter in pixel  
units.

**Example:**

```
Locate 20, 15
```

**See also:**

MOVETO

## **LOG()**

**Usage:**

$\text{Log}(x)$

x:                      Numeric value

**Description:**

LOG() returns the natural logarithm of the parameter  $x$ . The  
parameter must be greater than zero.

## **LPRINT**

**Usage:**

Lprint *Data* [,*Data*...]

Data: Variable or constant

**Description:**

LPRINT works like PRINT, but outputs *Data* to the printer instead of the screen. The output is buffered until a formfeed (ASCII character 12) is sent by the program. A formfeed can be sent by the command

```
Lprint Chr$(12)
```

The output is then printed. During the printing a message box is shown to allow the user to cancel the output. Printing data to the printer can be a very lengthy operation, especially when printing graphics.

**See also:**

PRINT

## ***LTRIM\$***

**Usage:**

Ltrim\$(*String\$*)

String\$: String

**Description:**

LTRIM\$( ) removes any leading blanks from *String\$*.



**Example:**

```
B$ = " 123"  
Print B$           'Result is: 123  
B$ = Ltrim$(B$)  
Print B$           'Result is: 123
```

**See also:**

TRIM\$()

## MAPMODE

**Usage:**

Mapmode *mode%* [, *G%*]

*mode%*: Integer  
*G%*: Graphics device number

**Description:**

MAPMODE determines the interpretation of coordinates on output devices. Possible values for *mode%* are:

- 1 TEXT mode
- 2 METRIC mode

In TEXT mode the unit of coordinates is a pixel of the screen or printer; in METRIC mode the unit is 0.1 mm. In METRIC mode exact output which is identical on different printers can be achieved with a resolution of 0.1 mm.

**Example:**

```
Mapmode 2          'METRIC 0.1mm
```

## **MENU**

**Usage:**

Menu *Str\$*

Str\$: String array

**Description:**

MENU creates a menu for a *WinBasic* program. *Str\$* is an array of strings, each of which contains the menu title and, optionally, the items in the menu. Each item in the string is separated by blanks.

**For example:**

```
Dim Menu$(3)
Menu$(1) = "File New Open Save"
```

This example defines a menu, titled File, containing three items, New, Open and Save.

The string may contain only one item; in that case, clicking the menu title generates a command. For example:

```
Menu$(2) = "Go!"
```

If one of the items in the menu needs to contain a blank, it can be entered as an underscore. Accelerators can be prefixed with an

ampersand, *e.g.* &File. Menu strings can contain tab characters (CHR\$(9)) for alignment, *e.g.* "Paste"+Chr\$(9)+"Shift+Ins".

The last item of *Str\$* must be an empty string. This terminates the list of menus. For example:

```
Menu$(3) = ""
```

When a menu item is selected by the user, a message is sent to the main program in the form of a numeric code which can be trapped by a Windows label. Read the section *Messages from Menus* in the first part of the manual for further details.

## ***MENUITEM***

### ***Usage:***

MenuItem *item%*, *onoff%*, *check%*

*item%*: Integer

*onoff%*: Integer

*check%*: Integer

### ***Description:***

MENUITEM allows an item in the current menu to be changed.

*item%* is the identifier of the item defined by the menu statement.

*onoff%* enables or disables a menu item:

0      Disable (grey) menu item

- 1 Enable menu item

All menu items are enabled by default. A disabled menu item appears greyed-out in the menu and cannot be selected by the user.

*check%* is used to place a check mark next to a menu item:

- 0 Clear check mark
- 1 Set check mark

**Example:**

```
MenuItem 1001, 0, 0      'disable 1. item in 1. menu
MenuItem 2003, 1, 1      'check 3. item in 2. menu
```

**See also:**

MENU

## MESSAGEBOX

**Usage:**

MessageBox *Text\$, Title\$, Button%, Answer%*

Text\$:           String  
Title\$:          String  
Button%:           Integer  
Answer%:       Integer variable

**Description:**

MESSAGEBOX displays a box with a message on the screen and waits for the user to press a button in response.

*Text\$* is the message in the box and *Title\$* the title of the box.

*Buttons%* defines the buttons displayed in the box:

- |   |                    |
|---|--------------------|
| 0 | OK                 |
| 1 | OK Cancel          |
| 2 | Abort Retry Ignore |
| 3 | Yes No Cancel      |
| 4 | Yes No             |
| 5 | Retry Cancel       |

The first button shown is the default push button. A different button can be the default push button by addition of a further value:

- |    |                          |
|----|--------------------------|
| 10 | Second button is default |
| 20 | Third button is default  |

The button pressed by the user is returned in *Answer%*:

- |   |        |
|---|--------|
| 1 | OK     |
| 2 | Cancel |
| 3 | Abort  |
| 4 | Retry  |
| 5 | Ignore |
| 6 | Yes    |
| 7 | No     |

**Example:**

```
f$ = "Test.dat"  
MessageBox "File not found, Create?", f$, 1+10, a%
```

## ***MID\$( )***

### **Usage:**

Mid\$(*String\$,Index%,Number%*)

String\$:	String
Index%:	Integer
Number%:	Integer

### **Description:**

MID\$( ) returns a partial string consisting of *Number%* characters from *String\$* beginning at position *Index%*. *String\$* remains unchanged. If *String\$* has less than *Number%* characters counted from *Index%* it is simply copied from *Index%* on.

### **Example:**

```
A$ = "ABcd"  
A$ = Mid$(A$,2,2)  
Print A$                                'Result is: Bc
```

### **See also:**

LEFT\$( ), RIGHT\$( )

## ***MOUSE***

**Usage:**

Mouse *Button%*, *x%*, *y%*

*Button%*: Integer  
*x%*, *y%*: Integers

**Description:**

The statement MOUSE should only be used after the Windows label \_MOUSE, which restarts program execution after the user has clicked a mouse button. The parameters *Button%*, *x%* and *y%* are instantiated with information about the pressed button and the current coordinates of the mouse (which should not be confused with the current coordinates for output of text or graphics).

*Button%* may contain one of the following:

- 1 Left button pressed
- 2 Right button pressed
- 3 Middle button pressed

**Example:**

```
_MOUSE:  
  Mouse button%, x%, y%  
  If button% = 1 Then MoveTo x%, y%
```

## MOVETO

**Usage:**

MoveTo *x%*, *y%* [*G%*]

*x%*, *y%*: Integers

G%:                Graphics device number

**Description:**

MOVETO sets the current coordinates of subsequent text and graphics output to the output device *G%* to *x%*, *y%*. MOVETO sets the coordinates in pixel units when in TEXT mode or units of 0.1 mm when in character units (see LOCATE).

**Example:**

see LINETO

**See also:**

LOCATE, LINETO

## OPEN

**Usage:**

Open *IO*, *Ident*, *Name\$* [,*Rec*]

IO:	String
Ident:	See below
Name\$:	String
Rec:	Record type

**Description:**

The file *Name\$* is opened for data I/O. *Name\$* can be a fully qualified filename including drive and directory.

*IO* is a one character string which defines the type of file access:



"I"	INPUT	Read only
"O"	OUTPUT	Write only
"U"	UPDATE	Read and Write
"A"	APPEND	Append to end of file
"R"	RANDOM	Random access

*Ident* is a file identifier which consists of the character '#' followed by an integer between 1 and 250. File operation statements reference the file by this identifier and not by the filename. As long as the file is open, no other file may be opened with the same identifier.

There must be a space before the file identifier; `OPEN#1` is a syntax error.

Files opened for input are read with the statements `INPUT` and `LINE INPUT`. Files opened for output are written with the statements `PRINT` or `WRITE`. The number of the characters read or written by one statement can vary with each execution. Files opened for update can be read and written with any of the statements mentioned above. The operations of reading and writing take place at the current location in the file, reached by previous actions.

If the end of a file opened for input or update is reached no further characters are read and a runtime error is generated which can be detected by the function `ERROR%( )`.

A file opened for input or update must already exist, otherwise a runtime error is generated. Opening a file for output

automatically creates a new file, erasing any existing file with the same name. A file opened for appending is created if it does not already exist.

When a random access file is opened, the recordtype *Rec* must be given to determine record length and structure of the file.

Runtime errors 1 and 4 may occur.

**Example:**

```
Rectype Arttype
...
Endrec
Open "R", #9 "article.dat", Arttype
Open "I", #1, "c:\MyDir\Fact.txt"
```

**See also:**

CLOSE

## **PIE**

**Usage:**

Pie *r%*, *beg%*, *end%* [*G%*]

*r%*: Integer  
*beg%*, *end%*: Integers  
*G%*: Graphics device number

**Description:**

PIE draws a pie consisting of an arc whose end points are

connected to the mid point by lines. The enclosed area can be filled with a hatch pattern. *r%* is the radius, *beg%* and *end%* are the start and end angles of the arc. The arc is drawn counterclockwise. The angles are measured against a downward pointing y-axis. If the start and end angle are equal, a filled circle is drawn. The mid point of the pie lies on the current coordinates (see MOVETO). The border of the pie is drawn in the current linestyle, pen width and colour. The enclosed area is filled with the currently selected fill pattern.

**Example:**

**See also:**

MOVETO, COLOUR, STYLE

## **PRINT**

**Usage:**

1. Print *Data* [,*Data...*]
2. Print *Ident*, *Data* [,*Data...*]

Ident: see OPEN

Data: Variable or constant

**Description:**

PRINT outputs data on the screen (first form) or to a file (second form).

Data is output in ASCII format. When the data items are separated by commas, the output data is aligned in columns with a width of 12 characters. If semicolons are used instead of commas, data items are written without spaces between them.

If there is a comma or semicolon after the last data item, the next PRINT command will continue printing at the current position, otherwise a carriage return is output, so that the next PRINT command begins on a new line.

The data items can be preceded by the word USING and a string template which formats the data before printing. The template can contain the characters "#", "." and ",". A decimal point is signified by ".", "#" is replaced by digits from *Data* and "," may delimit separate digits for improved readability. The template is valid for every *Data* that follows until a new template is used or the template is cancelled with USING "".

Runtime error 2 can occur.

**Example:**

```
Print "123";"456","789"
```

```
A = 1000.0
```

```
Print Using"##,###.##";A      'Result is: 1,000.00
```

**See also:**

OPEN, WRITE, LPRINT

## **PUT**

**Usage:**

Put *Ident*, *Recnr%*, *Rec*

Ident: See OPEN

Recnr%: Integer

Rec: Record

**Description:**

PUT writes records of fixed length to a random access file. The file must have been opened by OPEN as a random access file. Data from the record *Rec* is stored in the file at record number *Recnr%*.

Run time errors 2 and 3 can occur.

**Example:**

```
Record Atype Arec  
Put #2, 999, Arec
```

**See also:**

INPUT, GET

## **RECORD**

**Usage:**

Record *Rtype Name* [,*Name...*]

Rtype: Name of a record type

Name: Name of the defined record

**Description:**

The RECORD statement declares an instance of record type *Rtype* with the name *Name*.

The individual variables of the record *Name* are accessed by the compound name *Name.Item*, where *Item* is the name of a record variable defined by the RECVAR statements that define the structure of the record type *Rtype*.

*Name* must not have a type suffix.

Several records may be declared by one RECORD statement, each name being separated by commas.

**Example:**

See RECTYPE

## **RECT**

**Usage:**

Rect *width%*, *height%* [, *G%*]

*width%*: Integer

*height%*: Integer

*G%*: Graphics device number

**Description:**

RECT draws a filled rectangle, *width%* is the horizontal size of the rectangle, *height%* the vertical size. The upper left corner of the rectangle lies on the current coordinates. The border of the rectangle is drawn in the current line style and pen width and with the current colours. The enclosed area is filled with the currently selected fill pattern.

**See also:**

COLOUR, STYLE

## **RECTYPE**

**Usage:**

Rectype *Name*

Name: Name of the defined recordtype

**Description:**

RECTYPE defines a compounded data type with name *Name*. This name is used by records declared with the RECORD statement. Records are composed of simple data types using one or more RECVAR statements. The definition of a record is terminated by the ENDREC statement.

*Name* must follow the convention for real, *e.g.* no type suffix.

**Example:**

```
Rectype Articletype
  Recvar Artnr$ 12
  Recvar Artname$ 40
```

```
Recvar Price 8:2
...
Endrec
```

**See also:**

RECVAR, ENDREC, RECORD

## RECVAR

**Usage:**

```
Recvar Name Len[:Dec] [,Name...]
```

**Description:**

RECVAR statements define the collection of simple data types that comprise a record type. One or more items may be defined per RECVAR statement, separated by commas. An ENDREC statement terminates the definition of a record type.

*Name* is a variable name of type real, integer or string. Long and double types are not allowed.

Reals and integers are stored in ASCII form within records. The length of these numbers can be variable, therefore the length must be defined with *Len*. For reals the number of decimal places within the number must additionally be defined with *Dec*. The number of digits before the decimal point is *Len-Dec*.

Strings also have a fixed length within records, again defined by *Len*. When a string is assigned to a string in a record, it is fitted to the length of the record's string either by padding the string with



blanks or by truncating it. A string being assigned from a record's string may therefore have trailing blanks; these may be removed by the TRIM\$( ) function.

**Example:**

```
Rectype Atype
  Recvar a% 6
  Recvar sum 8:2
  Recvar str$ 12
Endrec

Record Atype Arec, Arecl

Arec.str$ = "abc"
a$ = Arec.str$
Print a$;"*"          'Result is: abc      *
```

**See also:**

RECORD

## REM

**Usage:**

Rem *Str*

Str:    Command

**Description:**

REM (short for REMark) introduces a comment into the program text.

A second form of remark is the apostroph ' which can stand anywhere in a line of program text to form a trailing remark after a statement

**Example:**

```
Rem This is a comment 'Example for REMARK
```

## ***RENAME***

**Usage:**

Rename *Old\$*, *New\$*

Old\$, New\$: Strings

**Description:**

RENAME renames a file. *Old\$* is the current name of the file, which may contain a drive and directory. *New\$* is the new name of the file, which may not contain a drive nor a directory as these remain unchanged.

**Example:**

```
Rename "c:\mydir\abc", "123"
```

**See also:**

## ***REPEAT ... UNTIL***

**Usage:**

Repeat

...

Until *Expression*

Expression:    Logical Expression

**Description:**

REPEAT starts a program loop which ends with an UNTIL statement. When the value of *Expression*, as checked by UNTIL, is false the loop is repeated, otherwise execution continues with the next statement after UNTIL.

In contrast to a WHILE loop, a REPEAT loop is always executed at least once, because the test for termination of the loop is at its end.

If a BREAK statement is executed in a REPEAT loop, program execution continues with the next statement after UNTIL.

In practice, *Expression* can be any expression, not just a logical one.

**Example:**

```
i% = 0
Repeat
  s% = s% + i%
  If s% > 100 Then Break 'leave loop
  Print "i%=";i%*10
  i% = i% + 1
Until i% > 20
```

**See also:**

BREAK

## ***RETURN***

### ***Usage:***

Return [*Value*]

Value: Variable or constant

### ***Description:***

RETURN terminates execution of a subroutine and optionally returns the value *Value* to the calling program. RETURN usually occurs at the end of the subroutine, but can occur anywhere between SUBROUTINE and ENDSUB. If a subroutine does not have a RETURN statement, it terminates execution at the ENDSUB statement.

By returning a value from a subroutine with RETURN, any one *WinBasic* subroutine can serve as a function.

The Type of *Value* must be same as the type of the subroutine.

RETURN is not allowed outside a subroutine.

### ***Example:***

```
Subroutine Sub2()  
...  
If ... Then Return 0.0  
...  
Return a
```

Endsub

**See also:**

SUBROUTINE

## ***RIGHT\$( )***

**Usage:**

Right\$(*String\$,Number%*)

String\$:               String

Number%:     Integer

**Description:**

RIGHT\$( ) returns a partial string consisting of the rightmost *Number%* characters from *String\$*. *String\$* remains unchanged. If *String\$* has less than *Number%* characters it is simply copied.

**Example:**

```
Print Right$("ABcd",2)   'Result is: cd
```

**See also:**

LEFT\$( ), MID\$( )

## ***SET\$***

**Usage:**

Set\$ *String\$, Index%, Partstring\$*

String\$: String variable  
Index%: Integer  
Partstring\$: String

**Description:**

SET\$ overwrites characters of *String\$* with characters from *Partstring\$*. *Index%* is the starting position in *String\$* for the first character of *Partstring\$*. If necessary, the length of *String\$* is increased to take all characters from *Partstring\$*.

**Example:**

```
A$ = "ABcd" : Part$ = "xx"  
Set$ A$, 2, Part$  
Print A$           'Result is: Axxd
```

**See also:**

INSERT\$

## SHOW

**Usage:**

Show *mode%* [,*Prog\$*]

mode%: Integer  
Prog\$: String

**Description:**

SHOW determines how the main window of program *Prog\$* is shown on the screen. *mode%* can have one of the following

values:

- 0 Hides the window, passing activation to another window
- 1 Activates and displays the window. If the window is minimized or maximized, it is restored to its original position
- 2 Activates the window and displays it as an icon.
- 3 Activates the window and displays it as a maximized window.
- 4 Displays a window in its most recent size and position. The window that is active remains active.
- 5 Activates a window and displays it in its current size and position.
- 6 Minimizes the window.
- 7 Displays the window as an icon. The window that is currently active remains active.

If *Prog\$* is omitted, the window of the running program is affected.

**Example:**

```
Show 7 'show own window as icon
Show 0, "clock.exe" 'hide clock
```

**See also:**

EXEC, SIZE

## ***SIN( )***

### ***Usage:***

Sin(*x*)

*x*:                      Numeric value

### ***Description:***

SIN( ) returns the sine of the parameter *x*, which must be given in radians.

## ***SIZE***

### ***Usage:***

Size *x%*, *y%*, *dx%*, *dy%* [,*Prog\$*]

*x%*, *y%*:                      Integers

*dx%*, *dy%*:                  Integers

*Prog\$*:                      String

### ***Description:***

SIZE defines the size and position of the window of program *Prog\$* or the window of the running program, if *Prog\$* is omitted.

*x%*, *y%* are the coordinates of the upper left corner, *dx%* and



$dy\%$  the horizontal and vertical size of the window, all measured in pixels.

**See *also*:**

EXEC, SHOW

## **SQR( )**

**Usage:**

Sqr(*x*)

*x*:                    Numeric value

**Description:**

SQR( ) returns the square root of the parameter *x*. The parameter must not be negative.

## **STATIC**

**Usage:**

Static *Name* [,*Name*...]

Name: Variable name

**Description:**

STATIC defines a variable in a subroutine as being static. In contrast to a LOCAL variable a STATIC variable is not volatile,

*i.e.* its value is preserved between calls of its subroutine.

At the start of program execution numeric STATICS are set to zero and string STATICS are set to "", as are global variables.

**Example:**

```
Static a22%, w#
```

**See also:**

LOCAL

## **STOP**

**Usage:**

```
Stop [Prog$]
```

Prog\$: String

**Description:**

STOP ends the execution of a program, if *Prog\$* is given, the program with that filename is stopped, otherwise the executing program stops itself.

**Example:**

```
Stop "Clock.exe"
```

**See also:**

END

# STR\$( )

## Usage:

Str\$(*number*)

number:                      Numeric constant or variable

## Description:

STR\$( ) converts the numeric value *number* into a string. In this way, an explicit type conversion of a number to a string is done.

## Example:

```
A$ = "***"+Str$(100.00)+"KG"
Print$                      'Result is: **100.000000KG
```

## See also:

VAL%( ), VAL( )

# STYLE

## Usage:

Style *linestyle%*, *fillpattern%*, *penwidth%*

linestyle%:    Integer  
fillpattern%:    Integer  
penwidth%:    Integer

## Description:

STYLE sets parameters for subsequent graphics output. Text output is not affected.

*linestyle%* sets the style of line for lines, arcs, ellipses and the borders of pies and rectangles:

0	Solid
1	Dashed
2	Dotted
3	Dash-Dot
4	Dash-Dot-Dot

The lines are drawn in the current foreground colour; gaps are filled with the current background colour.

*fillpattern%* selects a hatch pattern for the filling of pies and rectangles:

0	Empty	
1	Horizontal lines	
2	Vertical lines	
3	Backward diagonal lines	\\
4	Forward diagonal lines	///
5	Hatching	
6	Diagonal hatching	
7	Solid	

The hatch lines are drawn in the current foreground colour, the gaps in between with the current background colour.

*penwidth%* sets the width of lines in multiples of the default line

width. Line widths greater than one are always solid.

**See also:**

COLOUR

## ***SUBROUTINE***

**Usage:**

Subroutine *Name*[[*[[VAR]**Par*[,...]]]]

Name: Name

VAR: Reserved word

Par: Variable name

**Description:**

SUBROUTINE is the first statement of a subroutine. All the following statements belong to that subroutine until the subroutine is ended by an ENDSUB statement.

*Name* is the name of the subroutine, which is used to call the subroutine. A subroutine can be called by a GOSUB statement or as a function in an arithmetic expression.

The name of the subroutine also defines the type of the subroutine, in a manner similar to the naming of variables. This is important for subroutines which are called as functions in arithmetic expressions and return a value with a RETURN statement; otherwise, it is meaningless.

If a subroutine has no parameters, *Name* can be followed by an optional pair of parentheses ( ). Otherwise, the "formal parameters" of the subroutine are declared between the parentheses, separated by commas. These are variables of any type (single data types, arrays or records) which are placeholders for the actual parameters which are passed when calling the subroutine. When the subroutine is called the number and kind of formal and actual parameters must be the same.

The number of dimensions of any array must be shown in parentheses. For example, if one of the formal parameters is a two-dimensional array with name `arr`, the formal parameter must be written as `arr( , )`. For records the reserved word `RECORD`, then the record type, then the name of the formal parameter must be given (see the example below).

If the subroutine has a simple data type as a formal parameter, it can be passed an item from an array of the same type as an actual parameter when called, but only if the formal parameter is not a `VAR` parameter (see below).

Formal parameters are treated like local variables (see `LOCAL`) and can be used like them within the subroutine. Simple data type are passed "by value", arrays and records "by reference".

"By value" means that, when calling the subroutine, a local copy of the actual parameter is stored in the location of the formal parameter. If the variable is changed only the copy is changed; the contents of the original variable remains the same.

"By reference" means that the address in memory of the data is

passed to the subroutine and changes to the local variables also affect the original variable. Arrays and records are passed by reference because they can occupy prohibitively large amounts of memory. Simple data types can be called by reference by preceding the formal parameter with reserved word VAR. These parameters may not then be passed constants or items from arrays!

**Example:**

```
Dim areal(10,10), a(100)
...
Subroutine Sub1%(a$, a(,), b%, Record Rtype R)
  Local i%, d%
  d% = 0
  For i% = 1 To b%
    d% = d% + a(i%,i%)    'type mix!
  Endfor
  Print "This is a message from Sub1"
  Return d%
Endsub

Gosub Sub1%(MyString$(32), areal, 5, R1)
Dgn1% = Sub1%("Function", areal, a(3), R1)
```

This example shows a subroutine of type integer. It has the properties of a functions, because it returns a value that can be used in arithmetic expressions.

**See also:**

GOSUB, RETURN, DECLARE

# ***TAN( )***

## **Usage:**

Tan(*x*)

*x*:                Numeric value

## **Description:**

TAN( ) returns the tangent of parameter *x*, which must be given in radians.

# ***TIMEDATE\$( )***

## **Usage:**

Timedate\$(*n*)

*n*:                Integer

## **Description:**

TIMEDATE\$( ) returns the current date or time. The parameter *n* selects the returned result:

- |   |                                    |
|---|------------------------------------|
| 1 | Time                               |
| 2 | Date in US format (mm/dd/yy)       |
| 3 | Date in European format (dd.mm.yy) |

## **Example:**

```
Print Timedate%(1)            'Result is: 17:24:12
Print Timedate%(2)            'Result is: 10/28/89
```



```
Print Timedate%(3)      'Result is: 28.10.89
```

## ***TIMER***

### ***Usage:***

Timer *n*

n:           Integer

### ***Description:***

TIMER activates the Windows timer facility. *n* is the amount of time in milliseconds to elapse until the message \_TIMER is send to a *WinBasic* program. After this, the timer is reset. If repeated timer messages are needed, the timer can be started new while processing the \_TIMER statement.

### ***Example:***

see program DEMO.BAS

## ***TRACE***

### ***Usage:***

1. Trace on
2. Trace off
3. Trace *var* [,*var...*]

var:   Variable

**Description:**

TRACE ON steps in single line mode through the program. Every line of the program text (not every statement, since a line can contain several statements) is executed separately, then the window of the *WinBasic* editor is brought to the top and the next line to execute is shown in reverse video. The messagebox can be used to start execution of the next line (OK button) or leave the single line mode (Cancel button).

TRACE OFF stops the single line mode.

The third form displays a maximum of six variables in a special window for inspection. The single line mode must be set ON beforehand otherwise the statement is not executed. All trace statements can only operate if the program was compiled and started with the **Debug** option.

**Example:**

```
Trace on
...
Trace k%, a$(k%)
```

## ***TRIM\$( )***

**Usage:**

Trim\$(*String\$*)

String\$:       String

**Description:**

TRIM\$( ) removes any trailing blanks from *String\$*.

**See also:**

LTRIM\$( )

## UCASE\$( )

**Usage:**

Ucase\$(*String\$*)

String\$:       String

**Description:**

UCASE\$( ) returns a string containing *String\$* converted to upper case characters. *String\$* remains unchanged.

**Example:**

```
Print Ucase$("ABcd")                   'Result is: ABCD
```

**See also:**

LCASE\$( )

## VAL%( )

**Usage:**

Val%(*String\$*)

String\$:       String

**Description:**

VAL%( ) returns *String\$* converted into an integer.

**Example:**

```
N% = 100 + VAL%(10)
Print N%           'Result is: 110
```

**See also:**

VAL( ), STR\$( )

## VAL( )

**Usage:**

Val(*String\$*)

String\$:       String

**Description:**

VAL( ) returns *String\$* converted into a real.

**Example:**

```
R = 100.0 + VAL("10.0")
Print R           'Result is: 110.000000
```

**See also:**

VAL%( ), STR\$( )

# WHILE ... WEND

## Usage:

While *Expression*

...

Wend

Expression:    Logical expression

## Description:

WHILE starts a program loop which is executed as long as the logical expression *Expression* is true. If *Expression* is false before the first pass through the loop, the loop is not executed and the execution of the program continues with the next statement after the end of the loop. The end of a loop is marked by a WEND statement.

The loop can be terminated independently of the value of *Expression* by using the BREAK statement. This causes execution to continue at the next statement after WEND.

In practice, *Expression* can be any expression, not just a logical one.

## Example:

```
i% = 0
While i% <= 20
  s% = s% + i%
  If s% > 100 Then Break 'leave loop
  Print "i%=";i%*10
  i% = i% + 1
```

Wend

**See also:**

BREAK

## **WINTODOS\$( )**

**Usage:**

WinToDos\$(*String\$*)

String\$:       String

**Description:**

WINTODOS\$( ) is the inverse function to DOSTOWIN\$( ). It converts a character string from the Windows ANSI to the DOS character set.

**See also:**

DOSTOWIN\$( )

## **WRITE**

**Usage:**

Write *Ident, Data* [,*Data...*]

Ident:   See OPEN

Data:   Variable or constant

**Description:**

Like the second form of PRINT, WRITE outputs data to files. However, commas and semicolons between data items do not tab to the next column but instead are written to the file. When reading the file the commas and semicolons can be interpreted as separators.

If there is no semicolon or colon at the end of the line, a new line is created, otherwise a comma is written and the next output command continues at the current position.

Strings are written within a pair of quotation marks, "", so that commas in the string are not interpreted as separators when reading the file.

Runtime error 2 may occur.

**Example:**

```
Write #2, A, A$, 12
```

**See also:**

INPUT; PRINT

