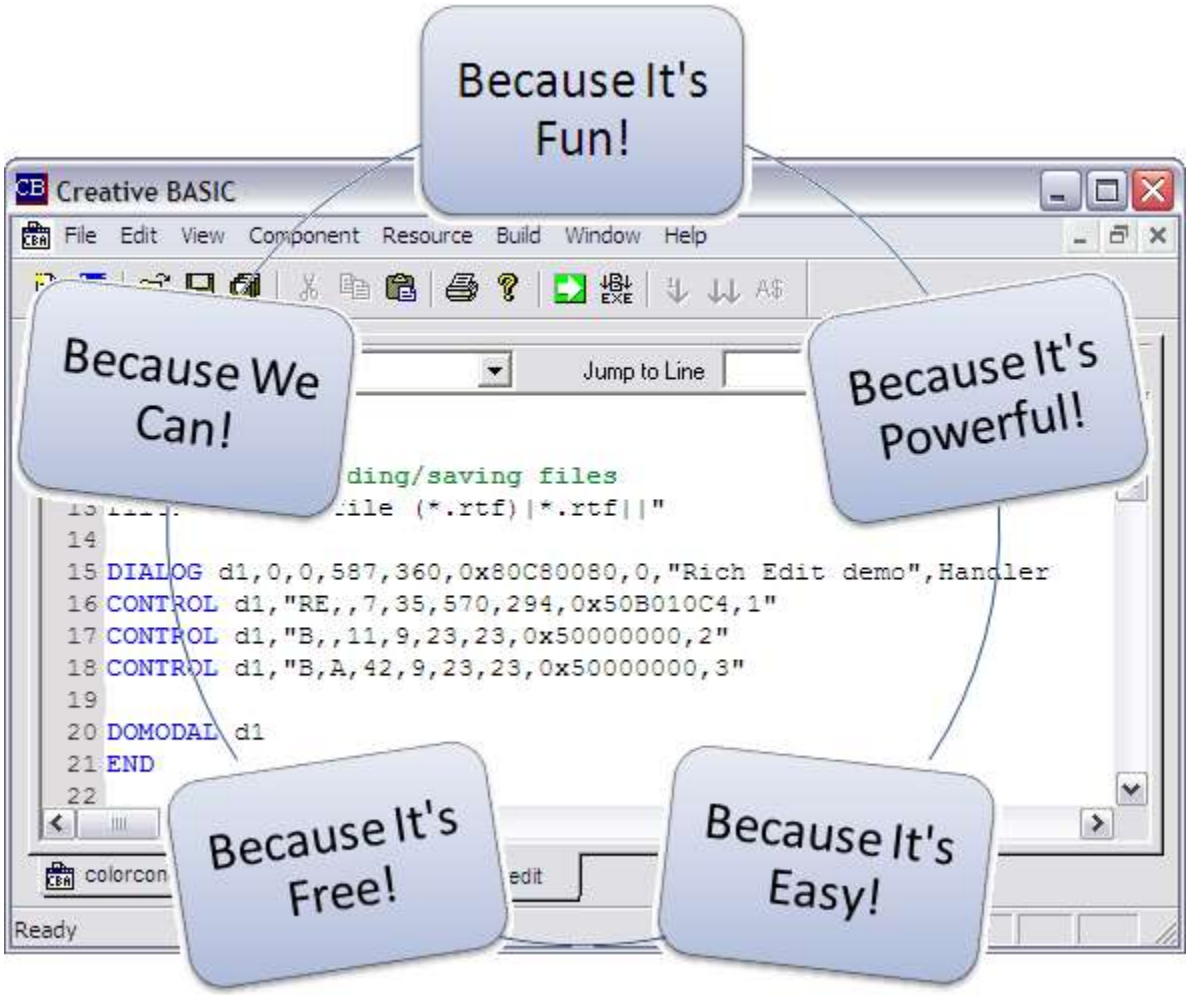
Creative BASIC



Command Dictionary

Contents

& (logical and Bitwise AND operator) 11
[] (array access) 11
: (define, newline) 12
' (apostrophe) 13
(division operator) 13
= (equality operator, assignment operator) 14
| (logical Exclusive OR operator) 15
. (dot) 16
> (more than) 16
<= (less than or equal to) 17
< (less than) 18
- (subtraction operator) 19
% (modulus operator) 19
>= (more than or equal to) 20
*** (multiplication operator)** 21
<> (not equal to) 21
| (logical and Bitwise OR operator) 22
() parenthesis 23
+ (addition operator, string concatenation) 24
^ (power of operator) 25
"" (string delimiter) 25
<> (not equal to) 25
ACOS() 26
ADDMENUITEM 27
ADDSTRING 28
ALIAS 29
ALLOCMEM 29
APPEND$ 30
ARRAYS 30
AS 31
ASC() 32
ASIN() 32
ATAN() 33

AUTODEFINE 33
ABS() 33
BROWSECMD 34
BACKPEN 36
CALL 36
CASE 37
CEIL() 39
CENTERWINDOW 39
CHECKMENUITEM 40
CHR$() 41
CIRCLE 41
CLEAR 42
CLOSECONSOLE 42
CLOSEDIALOG 43
CLOSEFILE 44
CLOSEPRINTER 45
CLOSEWINDOW 46
CLS 46
COLOR 47
COLORREQUEST() 48
CONST 49
CONTEXTMENU 50
CONTROL 51
CONTROLCMD() 52
CONTROLEXISTS() 54
COPYFILE() 55
COS() 56
COSH() 56
CREATE3DSCREEN 57
CREATEDIR 60
CREATESCREEN 61
D3DCAMERA() 62
D3DCOMMAND() 63
D3DDELETE 65

D3DLIGHT() 65
D3DMOVE 67
D3DRENDER 68
D3DSCENE() 69
D3DSETQUALITY 70
D3DSETRENDERMODE 72
D3DSHAPE() 73
DATE$ 74
DECLARE 75
DEFAULT 76
DELETE 77
DELETEFILE 77
DELETEIMAGE 77
DELETESTRING 79
DIALOG 80
DEF 81
DO 82
DOMODAL() 82
DRAWMODE 84
DXCREATEMAP 85
DXDRAWALLSPRITES 86
DXDRAWALLSPRITES 87
DXDRAWMAP 89
DXDRAWSPRITE 90
DXFILL 91
DXFLIP 92
DXGETMAPCOUNT 93
DXGETMAPHEIGHT() 95
DXGETMAPTILE() 96
DXGETMAPWIDTH() 98
DXGETSPRITEDATA 99
DXHITANY( ) 101
DXHITSPRITE( ) 102
DXHITSPRITETILE( ) 103

DXLOADMAP 105
DXLOADPALETTE 107
DXMOVEMAP 108
DXMOVESPRITE 110
DXNEWMAP 111
DXREMOVESPRITE 112
DXSAVEMAP 114
DXSCROLLMAP 115
DXSETCOLOR 117
DXSETMAPTILE 118
DXSETSPRITEDATA 119
DXSPRITE 121
ELLIPSE 122
ELSE 123
ENABLECONTROL 124
ENABLEMENU 125
ENABLECONTROL 126
ENABLETABS 127
END 128
ENDIF 129
ENDPAGE 129
ENDSELECT 130
ENDTYPE 131
ENDWHILE 132
EOF() 133
EXP() 133
FILEREQUEST() 134
FINDCLOSE() 135
FINDNEXT() 135
FINDOPEN() 136
FLOODFILL 137
FLOOR() 138
FONTREQUEST 139
FOR 140

FREELIB 141
FREEMEM 142
FRONTPEN 142
GET 143
GETBITMAPSIZE 144
GETCAPTION() 145
GETCARETPOSITION 146
GETCLIENTSIZE 147
GETCONTROLTEXT 148
GETDEFAULTPRINTER 149
GETDXVERSION 150
GETHDC() 150
GETKEYSTATE() 151
GETPIXEL() 151
GETPOSITION 152
GETSCREENSIZE 153
GETSCROLLPOS() 154
GETSCROLLRANGE 155
GETSELECTED() 156
GETSIZE 157
GETSTARTPATH 158
GETSTATE() 158
GETSTRING() 159
GETSTRINGCOUNT() 160
GETTEXTSIZE 161
GETUSERDATA 162
GOSUB 163
GOTO 164
HEX$() 164
IF 165
INKEY$ 166
INPUT 167
INSERTMENU 167
INSERTSTRING 168

INSTR 169
INT 170
ISARRAY 170
ISSELECTED 170
JUMP 171
LABEL 172
LCASE$() 172
LEFT$() 173
LEN() 173
LINE 174
LOADIMAGE() 175
LOADMENU() 176
LOADRESOURCE() 177
LOADTOOLBAR 178
LOCATE 179
LOG() 180
LOG10() 180
LTRIM$() 180
MENU 181
MESSAGEBOX 181
MID$() 182
MOVE 183
NEW() 183
NEXT 183
NOT 184
OPENCONSOLE 184
OPENFILE 185
OPENPRINTER() 185
PLAYWAVE 186
PRINT 186
PRINTWINDOW 187
PRTDIALOG() 188
PSET 188
PUT 189

RASTERMODE 190
READ() 191
READMEM 192
RECT 193
RELEASEHDC 193
REM 194
REMOVEDIR 195
REMOVEMENUITEM 195
REPLACE$ 196
RETURN 197
RGB() 197
RIGHT$() 198
RND() 198
RTRIM$() 199
SEEK 199
SELECT 200
SENDMESSAGE() 202
SETCAPTION 202
SETCONTROLCOLOR 203
SETCONTROLTEXT 203
SETCURSOR 204
SETFOCUS 205
SETFONT 206
SETHORIZEXTENT 207
SETICON 207
SETID 208
SETLBCOLWIDTH 209
SETLINESTYLE 209
SETPRECISION 210
SETSCROLLPOS 211
SETSCROLLRANGE 212
SETSELECTED 212
SETSIZE 213
SETSTATE 214

SETUSERDATA 215
SETUSERDATA 216
SETWINDOWCOLOR 217
SGN() 217
SHOWDIALOG 218
SHOWIMAGE 219
SHOWWINDOW 220
SIN() 220
SINH() 221
SPACE() 221
SQRT() 222
STARTTIMER 222
STEP 223
STOPTIMER 224
STR$() 225
STRING() 226
SUB 226
SYSTEM 227
TAN() 227
TANH() 228
TIME$ 228
TIMER 229
TO 229
TYPE 230
TYPEOF() 231
UCASE$() 231
UNTIL 232
USING() 233
VAL() 233
WAIT 234
WAITUNTIL 234
WHILE 235
WINDOW 236
WRITE() 237

WRITEMEM 238
WRITEPRINTER 239

& (logical and Bitwise AND operator)

Usage

```
expression & expression
```

Parameters

`expression` any expression which will evaluate to TRUE (1) or FALSE(0).

Returns

Nothing.

Description

Used to perform a logical AND operation on two expressions. Logical AND will result in a TRUE value only when both expressions are true. Logical AND will result in one of the results in the following table, depending on the values of the two expressions. The & operator can also be used to compute a bitwise AND operation on two numbers. Bitwise AND converts expression1 and expression2 to 32-bit unsigned integers, and returns a 1 in each bit position where the corresponding bits of both expression1 and expression2 are true. This new binary number is converted to decimal and returned.

expression1	expression2	OR result
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

Examples

```
OPENCONSOLE
PRINT "------------LOGICAL AND TABLE------------"
PRINT "expression1     expression2     OR result"
PRINT "   FALSE           FALSE           FALSE"
PRINT "   FALSE           TRUE            FALSE"
PRINT "   TRUE            FALSE           FALSE"
PRINT "   TRUE            TRUE            TRUE"
PRINT
PRINT "--------With numeric expressions---------"
PRINT "  (4 < 4)         (5 > 6)          FALSE"
PRINT "  (4 < 4)         (5 < 6)          FALSE"
PRINT "  (4 = 4)         (5 > 6)          FALSE"
PRINT "  (4 = 4)         (5 < 6)          TRUE"
PRINT
PRINT "Bitwise AND"
PRINT "first value = 11, which is 1011 in binary"
PRINT "first value = 7, which is 0111 in binary"
PRINT "11&7 (bitwise) = ",11&7
PRINT "Performing AND on 11 (1011) and 7 (0111) results in 3 (0011)"
PRINT
PRINT "Press any key to close the program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program prints out the basic logical AND truth table, and then completes the logical or table with some numeric expressions. It also works through a bitwise AND operation.

[] (array access)

Usage

```
variable[n]
```

Parameters

`variable` a variable which has been declared as an array.

`n` The index of the required element in the array.

Returns
The value contained in the specified element.
Description
This operator is both used to define and access an array.
Examples
```
OPENCONSOLE
DEF myArray[10]:INT  :'arrays are 0 based, so this array has element numbers 0 to 9
DEF loopcount : INT
FOR loopcount = 0 to 9
    INPUT "Enter a number to be stored:",myArray[loopcount]
    PRINT "That number was stored in element ", STR$(loopcount)
NEXT loopcount
PRINT
PRINT "The contents of the array myArray are as follow:"
FOR loopcount = 0 to 9
    PRINT "myArray[",STR$(loopcount),"]= ", STR$(myArray[loopcount])
NEXT loopcount
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program stores some values in an array and prints them out to screen.
See Also
ARRAY

: (define, newline)
Usage
```
DEF variablename : variabletype

-----OR-----

CODE : CODE
```
Parameters
variablename	any valid name for a variable.
variabletype	one of the standard or user - defined variable types.
code	any valid code.

Returns
Nothing.
Description
Depending on where it's used the : operator can either be used in the DEF statement in place of the AS command, or in some code to define a new line (required for inline comments).
Examples
```
OPENCONSOLE
'the : can be used to define a variable
DEF myvariable : INT
PRINT "Press any key to close this program"
DO :'The : on this line takes a new line, allowing you to put in a comment
UNTIL INKEY$ <> ""
```

```
CLOSECONSOLE
END
```

The above program code simply shows where you can use the : operator.

See Also

AS; DEF

' (apostrophe)

Usage

```
' comment
```

Parameters

comment — ' is followed by the text content of the comment.

Returns

Nothing.

Description

Signifies that the following txt is a comment. This is interchangeable with REM (remark). For inline comments (comments that appear on the same line as some code) you must use the newline operator (:) first.

Examples

```
OPENCONSOLE
'This is just a comment and will not effect the program at all
'Comments allow the programmer to include notes within the code
'This allows others to find out what your code does easily
PRINT "This is just a simple program that prints this text"
PRINT "Press any key to close this proram" :'This is an inline comment

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The aboveprogrm contains a few comments, to show how and why to use comments. When run, it simply prints out a line of text.

See Also

REM; : (colon)

/ (division operator)

Usage

```
number / number
```

Parameters

number — a numeric value.

Returns

Nothing.

Description

Used to perform mathematical division between numbers.

Examples

```
OPENCONSOLE
PRINT "3 / 3= ",3/3
PRINT "2 / 4= ",2/4
PRINT "5 / 10= ",5/10
PRINT
PRINT "Press any key to close this program"
```

```
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program displays a few examples of the division operator.

= (equality operator, assignment operator)

Usage

```
variable = value

-----OR-----

value = value
```

Parameters

`variable`	a variable.
`value`	any value.

Returns

Nothing.

Description

Can be used to assign a new value to a variable or to test if two values are equal.

Examples

```
OPENCONSOLE
DEF myvariable : INT
myvariable = 3
PRINT "Initial value of myvariable is ",myvariable
myvariable = 98
PRINT "After assigning a new value to myvariable, it's value is now ",myvariable
PRINT
IF 3 = 4
    PRINT "3 is equal to 4"
ELSE
    PRINT "3 is not equal to 4"
ENDIF
PRINT
IF 6 = 6
    PRINT "6 is equal to 6"
ELSE
    PRINT "6 is not equal to 6"
ENDIF
PRINT
IF "a" > "A"
    PRINT "a is equal to A"
ELSE
    PRINT "a is not equal to A"
ENDIF
PRINT
```

```
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program shows some examples of how to use the = operator. As well as comparing values, it can be used to compare variables, or to assign the value of one variable to another variale.

| (logical Exclusive OR operator)

Usage

```
expression | expression
```

Parameters

`expression` any expression which will evaluate to TRUE (1) or FALSE(0).

Returns

Nothing.

Description

Used to perform a logical XOR operation on two expressions. Logical XOR will result in a TRUE value unless both expressions are the same. Logical XOR will result in one of the results in the following table, depending on the values of the two expressions.

expression1	expression2	OR result
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

Examples

```
OPENCONSOLE
PRINT "-------------LOGICAL OR TABLE------------"
PRINT "expression1     expression2     OR result"
PRINT "   FALSE           FALSE           FALSE"
PRINT "   FALSE           TRUE            TRUE"
PRINT "   TRUE            FALSE           TRUE"
PRINT "   TRUE            TRUE            FALSE"
PRINT
PRINT "--------With numeric expressions---------"
PRINT "  (4 < 4)         (5 > 6)          FALSE"
PRINT "  (4 < 4)         (5 < 6)          TRUE"
PRINT "  (4 = 4)         (5 > 6)          TRUE"
PRINT "  (4 = 4)         (5 < 6)          FALSE"
PRINT
PRINT "Press any key to close the program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program prints out the basic logical OR truth table, and then completes the logical or table with some numeric expressions.

. (dot)

Usage

```
typevariable.membervariable
```

Parameters

`typevariable` a variable defined as a user type.
`membervariable` one of the variables contained in the user type.

Returns

Nothing.

Description

The dot opertor is used to access each of the member variables held in a user defined type.

Examples

```
OPENCONSOLE
TYPE contactdetails
   DEF Name:STRING
   DEF Age:INT
   DEF Phone[20]:ISTRING
ENDTYPE
DEF myContact : contactdetails
'Once a user type is defined, it can be used like all other inbuilt variable types
myContact.Name = "Joe Bloggs"
myContact.Age = 24
myContact.Phone = "01234567890"
PRINT myContact.Name
PRINT myContact.Age
PRINT myContact.Phone
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program simply shows how you can use the dot operator.

See Also

TYPE

> (more than)

Usage

```
value > value
```

Parameters

`value` any number or string value.

Returns

Nothing.

Description

Used to test if two numbers or strings are more than each other. This operator can be used in an IF or CASE statement. When comparing strings, this operator uses the ASCII values of the individual letters to compare them, therefore all lowercase letters will have a higher value than capital letters.

Examples

```
OPENCONSOLE
IF 3 > 4
    PRINT "3 is more than 4"
ELSE
    PRINT "3 is not more than 4"
ENDIF
PRINT
IF 6 > 5
    PRINT "6 is more than 5"
ELSE
    PRINT "6 is not more than 5"
ENDIF
PRINT
IF "a" > "A"
    PRINT "a is more than A"
ELSE
    PRINT "a is not more than A"
ENDIF
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above sample shows some uses of the > operator.

See Also

IF; CASE

<= (less than or equal to)

Usage

```
value <= value
```

Parameters

`value` any number or string value.

Returns

Nothing.

Description

Used to test if two numbers or strings are less than or equal to each other. This operator can be used in an IF or CASE statement. When comparing strings, this operator uses the ASCII values of the individual letters to compare them, therefore all lowercase letters will have a higher value than capital letters.

Examples

```
OPENCONSOLE
IF 3 <= 4
    PRINT "3 is less than or equal to 4"
```

```
ENDIF
PRINT
IF 6<= 5
    PRINT "6 is less than or equal to 5"
ELSE
    PRINT "6 is not less than or equal to 5"
ENDIF
PRINT
IF "a"<= "A"
    PRINT "a is less than or equal to A"
ELSE
    PRINT "a is not less than or equal to A"
ENDIF
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above sample shows some uses of the <= operator.

See Also

IF; CASE

< (less than)

Usage

```
value < value
```

Parameters

`value` any number or string value.

Returns

Nothing

Description

Used to test if two numbers or strings are less thaneach other. This operator can be used in an `IF` or `CASE` statement. When comparing strings, this operator uses the ASCII values of the individual letters to compare them, therefore all lowercase letters will have a higher value than capital letters.

Examples

```
OPENCONSOLE
IF 3 < 4
    PRINT "3 is less than 4"
ELSE
    PRINT "3 is not less than 4"
ENDIF
PRINT
IF 6 < 5
    PRINT "6 is less than 5"
ELSE
    PRINT "6 is not less than 5"
ENDIF
```

```
PRINT
IF "a" < "A"
    PRINT "a is less than A"
ELSE
    PRINT "a is not less than A"
ENDIF
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above sample shows some uses of the < operator

See Also

IF; CASE

- (subtraction operator)

Usage

```
number - number
```

Parameters

`number` a number.

Returns

Nothing.

Description

Used to perform mathematical subtraction, or to denote a number as being a negative (minus) number.

Examples

```
OPENCONSOLE
PRINT "3 - 3= ",3-3
PRINT "2 - 4= ",2-4
PRINT "5 - 10= ",5-10
PRINT "10 - -10=",10--10
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program displays a few examples of the - operator

% (modulus operator)

Usage

```
number % number
```

Parameters

`number` a number.

Returns

Nothing.

Description

Used to perform the mathematical modulus operation between numbers. Modulus returns the integer remainder when two numbers are divided.

Examples

```
OPENCONSOLE
PRINT "7 % 3= ",7%3
PRINT "4 % 2= ",4%2
PRINT "13 % 7= ",13%7
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program displays a few examples of the modulus operator.

>= (more than or equal to)

Usage

```
value >= value
```

Parameters

`value` any number or string value.

Returns

Nothing.

Description

Used to test if two numbers or strings are more than or equal to each other. This operator can be used in an `IF` or `CASE` statement. When comparing strings, this operator uses the ASCII values of the individual letters to compare them, therefore all lowercase letters will have a higher value than capital letters.

Examples

```
OPENCONSOLE
IF 3 >= 4
    PRINT "3 is more than or equal to 4"
ELSE
    PRINT "3 is not more than or equal to 4"
ENDIF
PRINT
IF 6>= 5
    PRINT "6 is more than or equal to 5"
ELSE
    PRINT "6 is not more than or equal to 5"
ENDIF
PRINT
IF "a" >= "A"
    PRINT "a is more than or equal to A"
ELSE
    PRINT "a is not more than or equal to A"
ENDIF
PRINT
```

```
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above sample shows some uses of the >= operator

See Also

IF; CASE

* (multiplication operator)

Usage

```
number * number
```

Parameters

`number` a number.

Returns

Nothing.

Description

Used to perform mathematical multiplication between numbers.

Examples

```
OPENCONSOLE
PRINT "3 * 3= ",3*3
PRINT "2 * 4= ",2*4
PRINT "5 * 10= ",5*10
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program displays a few examples of the multiplication operator.

<> (not equal to)

Usage

```
value <> value
```

Parameters

`value` any number or string value.

Returns

Nothing.

Description

Used to test if two numbers or strings are not equal to each other. This operator can be used in an `IF` or `CASE` statement. When comparing strings, this operator uses the ASCII values of the individual letters to compare them, therefore all lowercase letters will have a higher value than capital letters.

Examples

```
OPENCONSOLE
IF 3 <> 4
    PRINT "3 is not equal to 4"
ELSE
    PRINT "3 is equal to 4"
ENDIF
```

```
PRINT
IF 6 <> 6
    PRINT "6 is not equal to 6"
ELSE
    PRINT "6 is equal to 6"
ENDIF
PRINT
IF "a" <> "A"
    PRINT "a is not equal to A"
ELSE
    PRINT "a is equal A"
ENDIF
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above sample shows some uses of the <> operator

See Also

IF; CASE

| (logical and Bitwise OR operator)

Usage

```
expression | expression
```

Parameters

`expression` any expression which will evaluate to TRUE (1) or FALSE(0).

Returns

Nothing.

Description

Used to perform a logical OR operation on two expressions. Logical OR will result in a TRUE value unless both expressions are false. Logical OR will result in one of the results in the following table, depending on the values of the two expressions. The | operator can also be used to compute a bitwise OR operation on two numbers. Bitwise OR converts expression1 and expression2 to 32-bit unsigned integers, and returns a 1 in each bit position where the corresponding bits of either expression1 OR expression2 are 1. This new binary number is converted to decimal and returned

expression1	expression2	OR result
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Examples

```
OPENCONSOLE
PRINT "-------------LOGICAL OR TABLE------------"
PRINT "expression1     expression2     OR result"
PRINT "   FALSE           FALSE           FALSE"
PRINT "   FALSE           TRUE            TRUE"
```

```
PRINT "   TRUE            FALSE           TRUE"
PRINT "   TRUE            TRUE            TRUE"
PRINT
PRINT "--------With numeric expressions---------"
PRINT "  (4 < 4)         (5 > 6)          FALSE"
PRINT "  (4 < 4)         (5 < 6)          TRUE"
PRINT "  (4 = 4)         (5 > 6)          TRUE"
PRINT "  (4 = 4)         (5 < 6)          TRUE"
PRINT
PRINT "Bitwise OR"
PRINT "first value = 11, which is 1011 in binary"
PRINT "first value = 7, which is 0111 in binary"
PRINT "11|7 (bitwise) = ",11|7
PRINT "Performing OR on 11 (1011) and 7 (0111) results in 15 (1111)"
PRINT
PRINT "Press any key to close the program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program prints out the basic logical OR truth table, and then completes the logical or table with some numeric expressions. It also works through a bitwise OR operation.

() parenthesis

Usage

```
returnvalue = functionname(parameters)

-----OR-----

(10 - 6) + (2 * 6)
```

Parameters

None.

Returns

Nothing.

Description

Can be used to group maths terms, or logical directives (and, not, or) together, overriding the order in which they are executed. Can also be usedto group function parameters together before passing them to the function. Using parenthesis with a function usually indicates that it will return a value.

Examples

```
OPENCONSOLE
DEF aNumber : INT
'example of using parenthesis to group maths terms
aNumber = (2 * 5) - (6 -2) :'is equal to 10 - 4
PRINT aNumber
'example of using parenthesis to pass variables to a function
PRINT "COS(",aNumber,") = ",COS(aNumber)
PRINT "Press a key to close this program"
DO
UNTIL INKEY$ <> ""
```

```
CLOSECONSOLE
END
```

The above code shows the different ways in which you can use () parenthesis.

+ (addition operator, string concatenation)

Usage

```
number + number

-----OR-----

myString = string1 + string2
```

Parameters

`number` a number or variable containing a number.
`string1,string2` a string literal or variable.

Returns

Nothing.

Description

Used to add 2 numbers together. Or to concatenate (join) two strings.

Examples

```
'-----Example 1-----
OPENCONSOLE
PRINT "3 + 3= ",3+3
PRINT "2 + 4= ",2+4
PRINT "5 + 10= ",5+10
PRINT
PRINT "Press any ky to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END

'-----Example 2-----
OPENCONSOLE
DEF a$,b$,c$,d$,result$:STRING
a$ = "this "
b$ = "is "
c$ = "a "
d$ = "long string"
result$ = a$ + b$ + c$ + d$
PRINT result$
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program prints out a few exampes of the addition operator (+) (Example 1) and uses the + operator to concatenate (join) some strings together into one string.

^ (power of operator)

Usage

```
x ^ y
```

Parameters

x	a number.
y	a number.

Returns

Nothing.

Description

Used to compute the expression x to the power of y.

Examples

```
OPENCONSOLE
PRINT "3 ^ 3= ",3^3
PRINT "2 ^ 4= ",2^4
PRINT "5 ^ 10= ",5^10
PRINT
PRINT "Press any ky to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program shows a few example of the ^ operator and prints out the results.

"" (string delimiter)

Usage

```
"string content"
```

Parameters

string content	the literal string content.

Returns

Nothing.

Description

Used to show that everything between quote marks is a literal string.

Examples

```
OPENCONSOLE
DEF myString : STRING
myString = "This is a literal string"
PRINT myString
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above piece of code shows how to use "" to mark a string.

<> (not equal to)

Usage

```
value <> value

-----OR-----

value ~ value
```

Parameters

`value` any number or string value.

Returns

Nothing.

Description

Used to test if two numbers or strings are not equal to each other. This operator can be used in an `IF` or `CASE` statement. When comparing strings, this operator uses the ASCII values of the individual letters to compare them, therefore all lowercase letters will have a higher value than capital letters.

Examples

```
OPENCONSOLE
IF 3 <> 4
    PRINT "3 is not equal to 4"
ELSE
    PRINT "3 is equal to 4"
ENDIF
PRINT
IF 6 <> 6
    PRINT "6 is not equal to 6"
ELSE
    PRINT "6 is equal to 6"
ENDIF
PRINT
IF "a" <> "A"
    PRINT "a is not equal to A"
ELSE
    PRINT "a is equal A"
ENDIF
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above sample shows some uses of the <> operator

See Also

`IF; CASE`

ACOS()

Usage

```
ACOS(n)
```

Parameters

`n` a number in radians from -1.0 to 1.0.

Returns

The arcosine of `n` in radians.

Description

Calculates and returns the arcsine of `n`. `n` is an angle in radians.

Examples

```
OPENCONSOLE
PRINT "ACOS(.3) = ",ACOS(.3)
PRINT "Press any key to end this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Returns the value 1.27.

See Also

COS()

ADDMENUITEM

Usage

```
ADDMENUITEM winhandle, position, menutext, attributes, itemid
```

Parameters

`winhandle`	user defined `WINDOW` or `DIALOG` variable.
`position`	the position of the menu. Position starts at 0 for the leftmost menu, increasing by 1 for each menu as you move right.
`menutext`	the text to be displayed on the menu item.
`attributes`	there are 2 predefined attributes for a new menu item: @MENUDISABLE - the menu item will be grayed out and not selectable. @MENUCHECK - shows a checkmark next to the menu item. If you don't want to use either of the above, you can use the number 0 to create a standard menu item.
`itemid`	the ID of the the new menu item. This value is passed to your main program when the menu is clicked.

Returns

Nothing.

Description

Used to add a new menu item to an existing menu at runtime.

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Demo",main
MENU myWindow, "T, File, 0, 0"
ADDMENUITEM myWindow, 0, "Quit", 0, 1
ADDMENUITEM myWindow, 0, "Save", 0, 6
ADDMENUITEM myWindow, 0, "Load", 0, 7
ADDMENUITEM myWindow, 0, "Options", 0, 11

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
```

```
END

SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above code creates a window, adds a "File" menu item to the window, then adds four new menu items (Quit, Save, Load, Options) to the bottom of the leftmost (position = 0) menu in the window. No coding has been enabled for these items, so clicking on them will do nothing.

See Also

MENU, INSERTMENU

ADDSTRING

Usage

```
ADDSTRING winhandle, controlID, string
```

Parameters

`winhandle`	user defined `WINDOW` or `DIALOG` variable.
`controlID`	the ID number of the control.
`string`	the text you want to add.

Returns

Nothing.

Description

Adds a string to a list box or combo box control. `string` is added to the end of the list unless sorting is specified in the style of the control.

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Demo",main
CONTROL myWindow,"M,,250,100,300,100,@CTCOMBODROPDOWN,1"
ADDSTRING myWindow,1,"Just been added to this combo box"

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

This code creates a new window, adds a combobox control to the window, and adds an option to the combobox.

See Also

SETSELECTED; GETSELECTED(); ISSELECTED(); GETSTRINGCOUNT(); GETSTRING(); INSERTSTRING; DELETESTRING

ALIAS

Usage

```
ALIAS
```

Parameters

None.

Returns

Nothing.

Description

Used with `DECLARE` to create an alias of a DLL function name

Examples

```
DEF myWindow:WINDOW

WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"ALIAS Sample",main

DECLARE "User32", myMessageBox ALIAS MessageBoxA
(wnd:window,text:string,title:string,flags:int),int

CENTERWINDOW myWindow

myMessageBox(myWindow, "This messagebox has been called using the ALIAS command",
"Information", 0)


run = 1

WAITUNTIL run = 0

CLOSEWINDOW myWindow

END


SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above example creates an alias for the MessageBoxA DLL function. You would then use the alias name (`myMessageBox` in this case) instead of the original function name. The program will use the alias for the function to display a simple messagebox.

See Also

DECLARE

ALLOCMEM

Usage

```
ALLOCMEM (variable, count, size)
```

Parameters

`variable`	must be defined as type `MEMORY` with the `DEF` statement.
`count`	is the number of items this memory block can hold.
`size`	is the size of the items. The `LEN` function can be used to find the size of a variable.

Returns

-1 if an error occurs

Description

Before memory can be read or written to if first needs to be allocated.
Allocating tells the system that a certain amount of memory will be used by the program and should

not be written to by any other programs. Allocate memory with the ALLOCMEM function.

The LEN statement can be used to determine the size of any variable type. If variable was already in use, then ALLOCMEM would re-allocate the memory to be the size and count specified.

Examples

```
OPENCONSOLE
DEF buffer: MEMORY
DEF myNumber: INT
ALLOCMEM buffer, 100, LEN(myNumber)
PRINT "I Have just allocated enough memory for 100 items"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above example declares the variable buffer to be of type memory, myNumber to be an integer, then allocates enough memory to store 100 items the same size as the variable myNumber.

See Also

LEN(); FREEMEM

APPEND$

Usage

```
APPEND$ (string1, string2{, string3...})
```

Parameters

string1, string2{, string3...}	where each is a value to be appended.

Returns

A string made from the combined parameters.

Description

Concatenates all the strings in the parameter list and returns the total string.

Examples

```
OPENCONSOLE
DEF myString:STRING
myString = APPEND$("this ","is ","a ","string")
PRINT myString
PRINT "Press any key to end this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above code combines each of the pieces of text into one string.
When the code is run, the value of A$ will be "this is a string"

ARRAYS

Usage

```
DEF myArray[10]:INT
```

Parameters

The number of elements in your array must be stated in square brackets "[]" after the variable name. To define arrays with more than one dimension, separate the numbers by a comma e.g. DEF

myArray[10,10,10] : INT would create a 3-dimensional array with 10 elements in each dimension

Returns

Nothing.

Description

An array is a collection of data of the same type. Arrays are referenced by their variable name and an index. Think of a book full of squared paper. Each row of squares could be a 1 dimensional array, where you can put data into each square. Each page would therefore represent a 2 dimensional array, where each page contains a number of rows, therefore a number of arrays. Finally the whole book would be a 3 dimensional array, since it contains a number of pages, each containing a number of rows.

Examples

```
OPENCONSOLE
DEF myArray[10]:INT  :'arrays are 0 based, so this array has element numbers 0 to 9
DEF loopcount : INT
FOR loopcount = 0 to 9
    INPUT "Enter a number to be stored:",myArray[loopcount]
    PRINT "That number was stored in element ", STR$(loopcount)
NEXT loopcount
PRINT
PRINT "The contents of the array myArray are as follow:"
FOR loopcount = 0 to 9
    PRINT "myArray[",STR$(loopcount),"]= ", STR$(myArray[loopcount])
NEXT loopcount
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program asks for 10 numbers to fill an array. It then shows the contents of this array. Note how the `FOR` loop can be used to fill and read from an array.

See Also

ISARRAY

AS

Usage

```
DEF variablename AS variabletype
```

Parameters

None.

Returns

Nothing.

Description

The `AS` statement is used as part of the `DEF` statement, and is used to declare the type of variable you want. The ":" operator can be used in place of `AS`.

Examples

```
OPENCONSOLE
DEF variable1 AS INT
DEF variable2 : INT

'Both of the above statements will create an INT variable

variable1 = 15
variable2 = 25
PRINT variable1
PRINT variable2
```

```
PRINT
PRINT "Press any key to close this program"


DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program creates 2 variables, one using the AS command, one using the ":" operator. Different values are assigned to these, and they are printed to the screen.

See Also

DEF

ASC()

Usage

```
ASC(userVar)
```

Parameters

`userVar`	either a single character or a string. If passed a string, this function will return the ASCII value of the first character in the string.

Returns

An integer value.

Description

Returns the ASCII value of the character or the first character in the string.

Examples

```
OPENCONSOLE
PRINT "ASC("A") = ", ASC("A")
PRINT "ASC("Apple") = ", ASC("Apple")
PRINT
PRINT "When using ASC on a string of characters, it returns the ASCII value of
thePRINT "first letter only."
PRINT
PRINT "Press any key to end this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

In the example, both uses of `ASC` return the value of 65 (the ASCII value of "A").

See Also

CHR

ASIN()

Usage

```
ASIN(n)
```

Parameters

`n`	a number in radians from -1.0 to 1.0.

Returns

The arcsine of `n` in radians.

Description

Calculates and returns the arcsine of `n`. `n` is an angle in radians.

Examples

```
OPENCONSOLE
PRINT "ASIN(.3) = ", ASIN(.3)
PRINT "Press any key to end this program"
```

```
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Returns the value 0.30.

See Also

SIN()

ATAN()

Usage

```
ATAN(n)
```

Parameters

`n` a number in radians from -1.0 to 1.0.

Returns

The arctangent of `n`, in radians.

Description

Calculates and returns the arctangent of `n`. `n` is an angle in radians.

Examples

```
OPENCONSOLE
PRINT "ATAN(.3) = ", ATAN(.3)
PRINT "Press any key to end this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Returns the value 0.29.

See Also

TAN()

AUTODEFINE

Usage

```
AUTODEFINE "ON" | "OFF"
```

Parameters

None.

Returns

Nothing.

Description

Controls whether Creative BASIC allows auto definition of variables by assignment. Turning this feature off will force you to declare variables before you use them.

Examples

```
AUTODEFINE "OFF"
```

Creative BASIC will allow variable autodefinition when this is set to on. This means that you can use variables that were not declared with the DEF statement. This is set to "ON" as default.

See Also

DEF

ABS()

Usage

```
ABS(n)
```

Parameters

n a number.

Returns

A positive number.

Description

Computes and returns an absolute, positive value for the number specified by the parameter n.

When passed a number, this function ONLY returns a positive number. See the example below.

Examples

```
'Example 1
OPENCONSOLE
PRINT "ABS(10) = ", ABS(10)
PRINT "Press any key to end this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END

'Example 2
OPENCONSOLE
PRINT "ABS(-10) = ", ABS(-10)
PRINT "Press any key to end this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Returned value for both the above examples is 10.

When using this function, remember that it will always return a positive value.

See Also

SGN()

BROWSECMD

Usage

```
BROWSECMD window, command {,data}
```

Parameters

winhandle — a user defined WINDOW or DIALOG variable.

command — one of the following commands.

Command	Details
@NAVIGATE	Returns nothing. Navigates to the specified web page or local file. Uses 1 string parameter.
@GOHOME	Returns nothing. Loads the default 'home' page
@GOBACK	Returns nothing. Moves back one page in the history list. No parameters
@GOFORWARD	Returns nothing. Moves ahead one page in the history list. No parameters
@BROWSESTOP	Returns nothing. Stops loading of the current document. No parameters

@REFRESH	Returns nothing. Refreshes the current document. No parameters
@GETTITLE	Returns the title of the current document. No parameters. Used as a function.
@BACKENABLED	Returns 1 if there is at least one page to @GOBACK to. 0 otherwise. No Parameters. Used as a function
@FORWARDENABLED	Returns 1 if there is at least one page to @GOFORWARD to. 0 otherwise. No Parameters. Used as a function.
@CANCELNAV	Returns nothing. Cancels navigation to the current page. Use in response to @IDBEFORENAV to limit navigation. No parameters.
@GETPOSTDATA	Returns a string containing the data from a form. Filled in before @IDBEFORENAV is sent. No Parameters. Function.
@GETHEADERS	Returns a string containing the headers sent. Filled in before @IDBEFORENAV is sent. No Parameters. Function.
@GETNAVURL	Returns a string containing the URL. Filled in before @IDBEFORENAV or @IDNAVCOMPLETE is sent. No Parameters. Function.
@BROWSELOAD	Returns nothing. Loads the browser with the contents of a string. 1 string parameter.
@BROWSEPRINT	Returns nothing. Prints the currently displayed document. No Parameters.

`data` any further information required by the chosen `command`.

Returns

See the list of commands above for return information.

Description

Controlling an embedded browser is done with `BROWSECMD`. `BROWSECMD` serves as both a statement and a function depending on the command issued.

Examples

```
DEF myWindow:WINDOW
DEF run:INT

WINDOW myWindow,0,0,640,480,@SIZE|@BROWSER|@NOAUTODRAW|@MINBOX|@MAXBOX,0,"Demo Browser
",main
BROWSECMD myWindow,@NAVIGATE,"http://www.ionicwind.com"

run = 1
WAITUNTIL run=0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
        CASE @IDCLOSEWINDOW
```

```
            run = 0
    ENDSELECT
RETURN
```

The above code opens the webpage "http://www.ionicwind.com" in the embedded browser window myWindow.

See Also

WINDOW

BACKPEN

Usage

```
BACKPEN window, color
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
color	the value for the color you want. This is easily set by using the RGB function (see example).

Returns

Nothing.

Description

Sets the background color for all drawing operations in winhandle to color.

Examples

```
DEF myWindow:WINDOW
DEF run:INT
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Demo",main
PRINT myWindow,"Test Text before backpen used.  "
BACKPEN myWindow, RGB(170,170,170)
PRINT myWindow,"Test Text after backpen used."

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
        CASE @IDCLOSEWINDOW
            run = 0
    ENDSELECT
RETURN
```

The above code will set the backpen color for myWindow equal to light grey.

See Also

RGB(); FRONTPEN

CALL

Usage

```
CALL functionname {, parameters}
```

Parameters

Dependant on function being called.

Returns

Dependant on function being called.

Description

Calls a previously declared DLL routine named by `functionname`, optionally passes a variable list of parameters to the routine and can return a result.
Function must have been declared with the `DECLARE` statement.

You can also call a function by omitting the `CALL` statement altogether.

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Demo",main
DECLARE "User32",MessageBoxA (wnd:window,text:string,title:string,flags:int),int
MessageBoxA (myWindow,"This messagebox wasn't activated using CALL","Information",0)
CALL MessageBoxA,myWindow,"This messagebox was activated using CALL","Information",0

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above code declares a DLL function which will display a message box. Note the difference in calling the function with and without the `CALL` statement (no brackets).

CASE

Usage

```
CASE condition
```

Parameters

`condition` any valid condition.

Returns

Nothing.

Description

Defines a test condition of a `SELECT` statement. If the condition is true then all lines between this `CASE` statement and the next `CASE`, `DEFAULT` or `ENDSELECT` are executed.

Examples

```
'EXAMPLE 1
OPENCONSOLE
DEF Choice$: STRING
PRINT "Press some keys, Q to quit"
Choice$ = INKEY$
DO
    DO
    Choice$ = INKEY$
```

```
    UNTIL Choice$ <> ""

    SELECT Choice$
    CASE "A"
    CASE "a"
        PRINT "You pressed A!!"
    CASE "Z"
    CASE "z"
        PRINT "Z is my favorite letter!"
    CASE "Q"
    CASE "q"
        CLOSECONSOLE
        END
    DEFAULT
        PRINT "You pressed the letter ",Choice$
    ENDSELECT
UNTIL Choice$ ="q" | Choice$ ="Q"



'EXAMPLE 2
OPENCONSOLE
DEF Choice: INT
PRINT
DO
INPUT "Enter a Number, 0 to end: ", Choice
IF Choice = 0
    CLOSECONSOLE
    END
ENDIF
SELECT 1
    CASE (Choice > 10)
        PRINT "number is greater than 10"
    CASE (Choice < 10)
        PRINT "Number is less than 10"
    CASE (Choice = 10)
        PRINT "Number is equal to 10"
    ENDSELECT
PRINT
UNTIL Choice = "0"
```

In the first example we test the choice against multiple conditions using only one SELECT statement. Your conditions can be as complex as needed, only the first TRUE condition will be executed so make sure the conditions are unique

The SELECT statement can also be used to test multiple conditions and execute the first TRUE case. To do this use SELECT 1 as the opening statement and code each CASE statement as a condition (must be in brackets). The second example above shows this

See Also
SELECT; DEFAULT

CEIL()
Usage
```
CEIL(n)
```
Parameters
`n` a number.

Returns
The smallest integer greater than or equal to `n`.

Description
Returns the ceiling of the specified number or expression. The ceiling of a number is the closest integer that is greater than or equal to the number.

Examples
```
OPENCONSOLE
PRINT "CEIL(5.5) = ",CEIL(5.5)
PRINT "Press any key to end this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above code returns 6

See Also
FLOOR

CENTERWINDOW
Usage
```
CENTERWINDOW winhandle
```
Parameters
`winhandle` user defined `WINDOW` or `DIALOG` variable.

Returns
Nothing.

Description
Positions the chosen `window` in the center of the users screen.

Examples
```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"This window is Centered ",main
CENTERWINDOW myWindow
run = 1

WAITUNTIL run = 0
CLOSEWINDOW myWindow
END
```

```
SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The window referenced by the myWindow variable will now appear centered on the screen

See Also

SETSIZE

CHECKMENUITEM

Usage

```
CHECKMENUITEM winhandle, menuitem, state
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
menuitem	the ID of the menu item you want to mark with the checkmark.
state	0 or 1. Use 0 to erase the checkmark or 1 to show the checkmark.

Returns

Nothing.

Description

Sets or resets the checkmark next to a menu item. Use 0 to erase the checkmark or 1 to show the checkmark.

Examples

```
DEF myWindow:WINDOW
DEF menuchecked:INT
menuchecked=0
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Check Menu Demo",main
MENU myWindow, "T, File, 0, 0"
ADDMENUITEM myWindow, 0, "Check Menu", 0, 0

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDMENUPICK
        IF menuchecked = 0
            CHECKMENUITEM myWindow, 0, 1
            menuchecked = 1
        ELSE
            CHECKMENUITEM myWindow, 0, 0
            menuchecked = 0
        ENDIF
    CASE @IDCLOSEWINDOW
        run = 0
```

```
ENDSELECT
RETURN
```

The above code creates a window with one menu on it. When you click on the menu, it switches between being checked and not checked.

CHR$()

Usage

```
CHR$(n)
```

Parameters

n	integer, the ASCII value of a character

Returns

A character.

Description

Returns the character represented by the n parameter. This is the opposite of the ASC function. This function is useful for creating characters that can not be normally typed on a keyboard.

Examples

```
OPENCONSOLE
PRINT "CHR$(65) = ",CHR$(65)
PRINT "Press any key to end this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above sample returns the character "A"

See Also

ASC$

CIRCLE

Usage

```
CIRCLE winhandle, x, y, radius {,outline {,fill}}
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
x	the x position of the starting point of the circle.
y	the y position of the starting point of the circle.
radius	the radius of the circle.
outline	optional. The color of the circle's outline (the current foreground color is the default).
fill	optional. The color of the circle's center (if this is omitted, the circle will not be filled).

Returns

Nothing.

Description

The CIRCLE statement is used to draw device independent circles. The circle will be drawn at the starting point specified with the radius specified. The circle will be drawn in the current foreground color unless a border color is specified. The circle may be filled with an optional fill color.

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Circle Demo",main
CIRCLE myWindow,250,75,50,RGB(0,255,0),RGB(0,0,255)
```

```
run = 1

WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

The above sample creates a new window, and draws a blue circle with a green outline on it.

See Also

LINE; RECT

CLEAR

Usage

```
CLEAR variable
```

Parameters

`variable` the name of the variable to be cleared.

Returns

Nothing.

Description

Clears and undefines the variable. The variable name can then be defined to another type. Will not work on window or dialog variables.

Examples

```
DEF myVariable : INT
OPENCONSOLE

myVariable = 10

PRINT myVariable
CLEAR myVariable
PRINT myVariable

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

This code will cause an undefined variable error, because we have tried to print a variable which was undefined by the `clear` statement.

CLOSECONSOLE

Usage

```
CLOSECONSOLE
```

Parameters

None.

Returns

Nothing.

Description

Closes the console window if open. If the console window is not open then this statement does nothing.

Examples

```
OPENCONSOLE
PRINT "This is an example of how to use CLOSECONSOLE"
PRINT
PRINT "This window will stay open until you press a key"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program will open a console window and keep it open until you press a key.

See Also

OPENCONSOLE

CLOSEDIALOG

Usage

```
CLOSEDIALOG dialog,returnval
```

Parameters

dialog	The name of the DIALOG variable used to store the window.
returnval	The value to return. You can use the predefined values of @IDOK and @IDCANCEL if applicable.

Returns

The value passed to the function as return.

Description

Closes a dialog previously opened with the DOMODAL command.

Returns a numeric value to the DOMODAL statement which is normally @IDOK or @IDCANCEL.

Examples

```
DEF myDialog:DIALOG
DEF myWindow:WINDOW
DEF result:INT
DEF answer:STRING

'Open our window and define a dialog
WINDOW myWindow,0,0,640,200,@SIZE,0,"Dialog Test",wndproc
DIALOG myDialog,0,0,100,100,@CAPTION|@SYSMENU,myWindow,"My Dialog",dialoghandler
CONTROL myDialog,"B,OK,25,75,50,20,@TABSTOP,1"
CONTROL myDialog,"E,,15,45,70,14,@TABSTOP,2"

'Show the dialog
result = DOMODAL myDialog

'Print the result
```

```
MOVE myWindow,4,20
IF result = @IDOK
    PRINT myWindow, answer
ELSE
    PRINT myWindow, "DIALOG canceled"
ENDIF

'Just wait for the window to be closed
run=1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

'Our window subroutine
SUB wndproc
SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN

'Our dialog subroutine
SUB dialoghandler
SELECT @CLASS
    CASE @IDCONTROL
    SELECT @CONTROLID
        CASE 1
            answer = GETCONTROLTEXT(myDialog, 2)
            CLOSEDIALOG myDialog,@IDOK
    ENDSELECT
'All controls should be initialized while processing the @IDINITDIALOG message
    CASE @IDINITDIALOG
        CENTERWINDOW myDialog
        SETCONTROLTEXT myDialog,2,"Yipee!"
ENDSELECT
RETURN
```

The above code opens a window with a dialog box. If you click the button on the dialog, the text in the textbox will be printed on the window. If you cancel the dialog, by clicking the x button in the top right, the text displayed in the window will read "*DIALOG Cancelled*".

See Also

DIALOG

CLOSEFILE

Usage

```
CLOSEFILE filehandle
```

Parameters

`filehandle` the name of a file which was opened using the `OPENFILE` command.

Returns

Nothing.

Description

Closes a file previously opened with the `OPENFILE` command.

Examples

```
OPENCONSOLE
DEF myfile: FILE
IF (OPENFILE (myfile,"C:\CREATIVE BASICTEST.TXT","W") = 0)
    WRITE myfile,"This is a test"
    CLOSEFILE myfile
    PRINT "File created successfully"
ELSE
    PRINT "File could not be created"
ENDIF
PRINT "Press Any Key To Close"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program attempts to open a file in the C:\ directory called CREATIVE BASICTEXT.TXT. It writes a line of text to the file, then closes the file using the `FILECLOSE` command.

See Also

`OPENFILE`; `READ`; `WRITE`

CLOSEPRINTER

Usage

```
CLOSEPRINTER printerhandle
```

Parameters

`printerhandle` a printer opened with the `OPENPRINTER` function.

Returns

Nothing.

Description

After you are finished using the printer use the `CLOSEPRINTER` statement. `CLOSEPRINTER` finishes the current page, closes the document and frees any memory used by the system. You cannot use the handle for the printer again unless you reopen the printer with `OPENPRINTER`.

Examples

```
DEF hPrt: INT
DEF data: STRING
DEF name: STRING
DEF pagefrom,pageto,copies,collate: INT
pagefrom = 1
pageto = 1
copies = 1
name = PRTDIALOG (0,pagefrom,pageto,copies,collate)
hPrt = OPENPRINTER (name,"Test Document","TEXT")
IF (hPrt)
```

```
    data = "This is a test of printing"
    data = data + chr$(13)
    data = data + "This is line 2"
    WRITEPRINTER hPrt,data
    CLOSEPRINTER hPrt
ENDIF
END
```

The above code opens a printer, sends some text to the printer for printing, then closes the printer.

See Also

OPENPRINTER; WRITEPRINTER

CLOSEWINDOW

Usage

```
CLOSEWINDOW winhandle
```

Parameters

`winhandle` user defined `WINDOW` variable.

Returns

Nothing.

Description

Closes the window. Make sure you close all of your open windows before your program exits.

Examples

```
DEF myWindow: WINDOW
WINDOW myWindow,0,0,350,350,@MINBOX|@MAXBOX|@SIZE,0,"Simple Window",main
PRINT myWindow,"Click on the X in the top right corner to close"
REM when myWindow = 0 the window has been closed
WAITUNTIL myWindow = 0
END
'---
REM every time there is a message for our window
REM the operating system will GOSUB here
SUB main
IF @CLASS = @IDCLOSEWINDOW
    REM closes the window and sets myWindow = 0
    CLOSEWINDOW myWindow
ENDIF
RETURN
```

The above code defines and opens a window. Some text is printed on the window. If you click on the x in the top right, the window is closed with the `CLOSEWINDOW` command.

See Also

WINDOW

CLS

Usage

```
CLS
```

Parameters

None.

Returns

Nothing.

Description

Clears the console window.

Examples

```
OPENCONSOLE
PRINT "Here is some text"
PRINT
PRINT "Press the "c" key to clear the screen"

DO
UNTIL INKEY$ = "c"
CLS
PRINT "All the other text has been cleared."
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

This code will print some text on the screen. If you press c it will clear the screen of all text, then press any key and the program will quit.

See Also

COLOR

COLOR

Usage

```
 COLOR forground,background
```

Parameters

`foreground` a number between 0 and 15 representing the color you want the text foreground to be.

`background` a number between 0 and 15 representing the color you want the text background to be.

COLOR number	Color Produced
0	BLACK
1	BLUE
2	GREEN
3	CYAN
4	RED
5	MAGENTA
6	BROWN
7	WHITE
8	GREY
9	LIGHT BLUE

10	LIGHT GREEN
11	LIGHT CYAN
12	LIGHT RED
13	LIGHT MAGENTA
14	YELLOW
15	HIGH INTENSITY WHITE

Returns
Nothing.
Description
This command is used to change the foreground color and background color of any subsequent text in the console window.
Examples

```
OPENCONSOLE
PRINT "This is the normal text color"
COLOR 4,7
PRINT "This is Red writing on a White Background"
COLOR 1,8
PRINT "This is Blue writing on a Grey Background"
COLOR 7,0
PRINT

PRINT "Press any key to end this program"
DO
UNTIL INKEY$ <>""
CLOSECONSOLE
END
```

The above program will change the text colors a few times, printing out some text to show this happening.
See Also
BACKPEN

COLORREQUEST()

Usage

```
COLORREQUEST( winhandle {, color} )
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
color	specifies a color to highlight when the dialog is displayed.

Returns
Returns the color selected or -1 if the user cancels the dialog.
Description
Opens the standard color palette dialog so the user can select a color.
Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Demo",main
```

```
FRONTPEN myWindow, COLORREQUEST (myWindow)
PRINT myWindow,"You have just chosen the color of this text"

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
        CASE @IDCLOSEWINDOW
            run = 0
    ENDSELECT
RETURN
```

This program will allow you to choose the color of the text before it is printed on the window.

See Also

RGB; FRONTPEN; BACKPEN

CONST

Usage

```
CONST name = expression|numeric
```

Parameters

None.

Returns

Nothing.

Description

Used to create a reference to a constant value. Constants cannot be modified once they are set. PI is a good example of this, since the word PI is easier to type than the number, and the number will never change.

Examples

```
OPENCONSOLE
DEF radius : FLOAT
DEF area : FLOAT
CONST pi = 3.14

INPUT "Enter the radius of a circle and press enter ", radius
area = pi * (radius * radius)
PRINT
PRINT "The area of a circle with radius ", radius, " is ", area
PRINT
PRINT "Press any key to end this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

This program uses a constant pi as a reference to the value 3.14. It then takes in a value for a circle's radius and outputs the area.

See Also

DEF

CONTEXTMENU

Usage

```
CONTEXTMENU winhandle, x, y, definition{, definition.}
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
x	the x coordinate of the top left corner of the menu.
y	the x coordinate of the top left corner of the menu.
definition	a string describing the menu item. the string is in the format: "[T|I|S], text, attributes, ID" T = Title I = Item S = Sub item (popup) text = the text to display on the menu item attributes = 0 for a normal menu, @MENUDISABLE for a greyed out menu, @MENUCHECK for a menu with a checkmark ID - the ID number of the menu item

Returns

Nothing.

Description

Displays a popup context menu in your window or dialog. This is normally done in response to a @IDRBUTTONUP message. x an y define the position in the window where the popup menu will be displayed.

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Context Menus Sample",main
CENTERWINDOW myWindow
run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDRBUTTONUP
        CONTEXTMENU myWindow, @MOUSEX, @MOUSEY, "I,Exit,0,99"
    CASE @IDMENUPICK
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

This program creates a menu when you right click on the open window. You can close this window by

choosing the exit option from the context menu. The @MOUSEX and @MOUSEY commands are system variables which show the x and y coordinate values of the mouse pointer.

See Also

MENU

CONTROL

Usage

```
CONTROL winhandle, definition{,definition..}
```

Parameters

winhandle — user defined WINDOW or DIALOG variable.

definition — a string literal or variable with the following syntax:
"type,title,x,y,width,height,style,ID"

type = one of the following values:

B	creates a button
E	creates an edit box
S	creates a scrollbar
R	creates a radio button
C	creates a check box
L	creates a list box
M	creates a combo box
T	creates a static text control
LV	creates a List View control
RE	creates a Rich Edit control
SW	creates a status window

title = the text to appear on the control.

x = the x coordinate of the top left hand corner of the control.

y = the y coordinate of the top left hand corner of the control.

width = the width of the control.

height = the height of the control.

style = a style for the control. 0 for no style. There are lots of styles for each control. Please refer to the section on Controls for a list of these.

ID = the ID number of the control.

Returns

Nothing.

Description

Adds a control to a window or dialog. The definition is a string literal or variable with the following syntax: "type,title,x,y,width,height,style,ID".

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Controls Sample",main
CONTROL myWindow,"B,Close Window, 4,100,100,20,0,2"
```

```
run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

Would create a BUTTON control in the window at position 4,100 with a width of 50 and a height of 20. No style is given and the button has an ID of 2.

CONTROLCMD()

Usage

```
CONTROLCMD ( winhandle, ID, command {,param...} )
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
controlID	the ID number of the control.
command	the command you want to perform (see the section on controls for more information on this).
parameters	the parameters required by the command (see the section on controls for more information on this).

Returns

Dependant on the command passed (see the section on controls for more information on this).

Description

Used to retrieve and set properties for controls.

Examples

```
'demo of the rich edit control
'Requires Creative BASIC 1.94 or greater
DEF d1:Dialog
DEF textclr,size,weight,bold,flags:INT
DEF fontname,oldfont,filename:STRING
DEF file1:FILE
'the default font properties
fontname = "Arial"
oldfont = fontname
size = 12
weight = 400

DIALOG d1,0,0,587,360,0x80C80080,0,"Rich Edit demo",Handler
CONTROL d1,"RE,,7,35,570,294,0x50B010C4,1"
CONTROL d1,"B,Color,10,9,40,23,0x50000000,2"
```

```
CONTROL d1,"B,Font,60,9,40,23,0x50000000,3"

DOMODAL d1
END

Handler:
SELECT @CLASS
    CASE @IDINITDIALOG
        MENU d1,"T,File,0,0","I,Quit,0,1"
        INSERTMENU
d1,1,"T,Edit,0,0","I,Undo,0,2","I,Cut,0,3","I,Copy,0,4","I,Paste,0,5"
        INSERTMENU d1,2,"T,Format,0,0","I,Subscript,0,8","I,Superscript,0,9","I,
        Normal,0,10"
        'default font and color
        CONTROLCMD d1,1,@RTSETDEFAULTFONT,fontname,size,0,0
        CONTROLCMD d1,1,@RTSETDEFAULTCOLOR,RGB(0,0,0)
        'margins
        CONTROLCMD d1,1,@RTSETMARGINS,10,0
        '7.5 inch line size
        CONTROLCMD d1,1,@RTSETLINEWIDTH,1440 * 7.5
        '128K text limit
        CONTROLCMD d1,1,@RTSETLIMITTEXT,128000
        SETCONTROLCOLOR d1,2,0,RGB(255,0,0)
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 2:'Color
                textclr = COLORREQUEST(d1,textclr)
                CONTROLCMD d1,1,@RTSETSELCOLOR,textclr
                SETFOCUS d1,1
            CASE 3:'FONT
                'flags = 0
                fontname = FONTREQUEST(d1,size,weight,flags,textclr,oldfont)
                if(len(fontname))
                    oldfont = fontname
                    if weight = 700 THEN bold = 1 ELSE bold = 0
                    CONTROLCMD d1,1,@RTSETSELFONT,fontname,size,bold,flags
                    CONTROLCMD d1,1,@RTSETSELCOLOR,textclr
                    SETFOCUS d1,1
                endif
        ENDSELECT
    CASE @IDMENUPICK
        SELECT @MENUNUM
            CASE 1:'Quit
                CLOSEDIALOG d1,@IDOK
            CASE 2:'Undo
```

```
                CONTROLCMD d1,1,@RTUNDO
            CASE 3:'Cut
                CONTROLCMD d1,1,@RTCUT
            CASE 4:'Copy
                CONTROLCMD d1,1,@RTCOPY
            CASE 5:'Paste
                CONTROLCMD d1,1,@RTPASTE
            CASE 8:'Subscript
                CONTROLCMD d1,1,@RTSETCHAROFFSET,-((size / 2) * 20)
            CASE 9:'Superscript
                CONTROLCMD d1,1,@RTSETCHAROFFSET,((size / 2) * 20)
            CASE 10:'normal
                CONTROLCMD d1,1,@RTSETCHAROFFSET,0
        ENDSELECT
    CASE @IDMENUINIT
        'only enable the Undo menu if there is something to..undo
        ENABLEMENUITEM d1,2,CONTROLCMD(d1,1, @RTCANUNDO)
ENDSELECT
RETURN
```

The above is a small wordpad-type program, which allows you to do some basic word processor functions. The above code uses `CONTROLCMD` functions for things like cut, paste and copy.

CONTROLEXISTS()

Usage

```
CONTROLEXISTS( winhandle, ID )
```

Parameters

`winhandle`	user defined `WINDOW` or `DIALOG` variable.
`ID`	the ID number of the control.

Returns

`INT`	1 if the control exists, 0 if it does not exist.

Description

This command checks to see if a control exists or not. Useful for controls that are dynamically created based on user options.

Examples

```
DEF myWindow:WINDOW
DEF loopcount, ypos : INT
ypos=50
WINDOW myWindow,0,0,640,500,@SIZE|@MINBOX|@MAXBOX,0,"CONTROLEXISTS Sample",main
CONTROL myWindow,"B,Close Window,50,50,200,20,0,1"
FOR loopcount = 1 to 10
IF (CONTROLEXISTS(myWindow,loopcount)=0)
    ypos = ypos + 30
    CONTROL myWindow,"B,Button Number " + str$(loopcount) + ",50," + str$(ypos) +
",200,20,0," + str$(loopcount)
ELSE
```

```
    MESSAGEBOX myWindow, "Control number " + str$(loopcount) + " already exists!",
"Information"
ENDIF
NEXT loopcount

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
    SELECT @CONTROLID
        CASE 1
            run = 0
        DEFAULT
            MESSAGEBOX myWindow, "You clicked on button number" + str$(@CONTROLID),
"Information"
    endselect
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above program creates 10 buttons dynamically, checking to see if each one already exists. Control ID 1 is the Close Window button at the top, so this program will skip the first button, and create 9 more.

COPYFILE()

Usage

```
COPYFILE( source, dest, overwrite )
```

Parameters

`source`	the source file you want to copy.
`dest`	the destination you want to copy the file to.
`overwrite`	if `overwrite` = 0 then the function will overwrite the file if it exists. If `overwrite` = 1 then the function will not overwrite the file if it exists.

Returns

`error` If an error occurs, 0 will be returned.

Description

Copies the file specified by the source path to the destination path. If fail = 1 then the function will not copy the file if it already exists in the destination directory. If fail = 0 then the file will be overwritten. COPYFILE returns 0 on error.

Examples

```
OPENCONSOLE
DEF myfile : FILE
IF(OPENFILE(myfile,"C:\copyfile.txt","W") = 0)
    WRITE myfile, "this is the source document"
```

```
    CLOSEFILE myfile
ENDIF

IF(COPYFILE("C:\copyfile.txt", "C:\copy of copyfile.txt", 0) = 0)
    PRINT "An error occurred in copying this file"
    PRINT
ELSE
    PRINT "File was copied successfully"
    PRINT
ENDIF
PRINT "Press any key to end this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program checks to see if a file named copyfile.txt is in your c:\ directory. If it is not, a file of that name will be created. The file is then copied to c:\copy of copyfile.txt.

See Also

DELETEFILE

COS()

Usage

```
COS(n)
```

Parameters

n — any numeric expression.

Returns

A number - the COSine of the numeric expression 'n'.

Description

Returns the COSine of the numeric expression 'n'.

Examples

```
OPENCONSOLE
PRINT "COS(5.6) = ", COS(5.6)
PRINT
PRINT "Press any key to end this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program returns 0.78.

See Also

COSH; SIN; TAN

COSH()

Usage

```
 COSH(n)
```

Parameters

n — a numeric expression.

Returns

A number.

Description

Returns the Hyperbolic COSine of the numeric expression 'n'.

Examples

```
OPENCONSOLE
PRINT "COSH(5.6) = ", COSH(5.6)
PRINT
PRINT "Press any key to end this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program returns 135.22.

See Also

TAN; COS; SIN; TANH; SINH

CREATE3DSCREEN

Usage

```
CREATE3DSCREEN winhandle, width, height {,bpp}
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
width	the width (in pixels) of the 3d screen you want to create.
height	the height (in pixels) of the 3d screen you want to create.
bpp	Bits Per Pixel. The color depth of the screen (i.e. 8bit, 16bit, 32bit). If you include this, a fullscreen window will open using the color depth specified. If you do not include this, a window of the specified width and height will be opened. This window will use the color settings currently in use by the PC it's running on.

Returns

A value greater than or equal to 0 on success.

Description

Creates a Direct3D screen and attaches it to the window specified. If bpp is specified the Direct3D screen will be full screen. Otherwise it will be a windowed surface.

Examples

```
DECLARE CreatePyramid(parent:INT,height:float)

IF GETDXVERSION < 7
    MESSAGEBOX 0,"This program requires" + chr$(13) + "DirectX 7.0 or greater","Error"
    END
ENDIF

DEF win:window
DEF scene,camera,shape,light:int

WINDOW win,0,0,640,480,@CAPTION|@NOAUTODRAW,0,"Direct3D Example 3",mainwindow
IF CREATE3DSCREEN(win,640,480) <> 0
    MESSAGEBOX win, "Could not create Direct3D screen","Error"
    CLOSEWINDOW win
    END
ENDIF
```

```
'Set the render quality for the entire scene.
'flat shading is sufficient since our pyramid has no curves.
D3DSETQUALITY win,@LIGHTON | @FILLSOLID | @SHADEFLAT
FRONTPEN win,RGB(255,255,255)
BACKPEN win,0
DRAWMODE win,@TRANSPARENT

'The parent scene frame
scene = D3DSCENE(win)
D3DCOMMAND scene,@SETSCENEBACKCOLOR,.5,.5,.5

'Create and position the camera
camera = D3DCAMERA(scene)
D3DCOMMAND camera,@SETPOSITION,scene,0,0,0
D3DCOMMAND camera,@SETORIENTATION,scene,0,0,1,0,1,0

'create and position a light source
light = D3DLIGHT(scene,@LIGHTAMBIENT,1.0,1.0,1.0)
D3DCOMMAND light,@SETPOSITION,scene,0,100,50
D3DCOMMAND light,@SETORIENTATION,scene,0,-1,0,0,1,0

'create a custom shape
shape = CreatePyramid(scene,7)
D3DCOMMAND shape,@SETPOSITION,scene,0,0,50
D3DCOMMAND shape,@SETROTATION,0,0,1,0,(-.7 * (3.1415/180))
'set each face of the pyramid to a different color
D3DCOMMAND shape,@SETFACECOLOR,0,0,0,1
D3DCOMMAND shape,@SETFACECOLOR,1,0,1,0
D3DCOMMAND shape,@SETFACECOLOR,2,1,0,0
D3DCOMMAND shape,@SETFACECOLOR,3,1,1,0

run=1
'start with a black background
SETWINDOWCOLOR win,RGB(0,0,0)
DXFILL win,RGB(0,0,0)
'process messages until somebody closes us
waituntil run=0

'delete all the frames
D3DDELETE light
D3DDELETE shape
D3DDELETE camera
D3DDELETE scene
```

```
closewindow win
end

SUB mainwindow
SELECT @class
    CASE @IDCHAR
        if (@CODE = ASC("Q")) | (@CODE = ASC("q")) THEN run = 0
    CASE @IDCREATE
        centerwindow win
    CASE @IDCLOSEWINDOW
        run=0
    CASE @IDDXUPDATE
        'Move the frames according to their current rotation, direction and velocity
        D3DMOVE win,1
        'The first camera can move in the scene using the arrow keys
        IF(GETKEYSTATE(0x26))
            D3DCOMMAND camera,@ADDTRANSLATION,0,0,.5
        ENDIF
        IF(GETKEYSTATE(0x28))
            D3DCOMMAND camera,@ADDTRANSLATION,0,0,-.5
        ENDIF
        IF(GETKEYSTATE(0x25))
            D3DCOMMAND camera,@ADDROTATION,0,1,0,(-.5 * (3.1415/180))
        ENDIF
        IF(GETKEYSTATE(0x27))
            D3DCOMMAND camera,@ADDROTATION,0,1,0,(.5 * (3.1415/180))
        ENDIF
        'render the scene to the DirectX surface
        D3DRENDER scene,camera
        'add any 2D elements after the scene is rendered.
        MOVE win,0,15
        PRINT win,"Move around the scene with the arrow keys"
        'show the DirectX surface
        DXFLIP win,0,0
ENDSELECT
RETURN


SUB CreatePyramid(parent:INT,height:float)
DEF pyr:INT
DEF facearray[38]:INT
    x = height/2
    y = height/2
    z = height/2
```

```
    pyr = D3DSHAPE(parent,@SHAPECUSTOM)

    'describe the vertexes
    D3DCOMMAND pyr,@ADDVERTEX,0,y,0
    D3DCOMMAND pyr,@ADDVERTEX,-x,-y,-z
    D3DCOMMAND pyr,@ADDVERTEX,x,-y,-z
    D3DCOMMAND pyr,@ADDVERTEX,-x,-y,z
    D3DCOMMAND pyr,@ADDVERTEX,x,-y,z

    'descibe the normal directional vectors
    'the vector describes what direction a face is pointing.
    D3DCOMMAND pyr,@ADDNORMAL,-1,1,0
    D3DCOMMAND pyr,@ADDNORMAL,1,1,0
    D3DCOMMAND pyr,@ADDNORMAL,0,1,-1
    D3DCOMMAND pyr,@ADDNORMAL,0,-1,1
    D3DCOMMAND pyr,@ADDNORMAL,0,-1,0

    'descibe the faces
    'format of the face array is
    ',,,...,,
    'n is the number of vertex/normal tuples. The last entry ends in 0
    'vertexes must be entered in clockwise direction to correctly describe the face.
    facearray =     3,3,0,0,0,1,0
    facearray[7] =  3,1,2,0,2,2,2
    facearray[14] = 3,2,1,0,1,4,1
    facearray[21] = 3,4,3,0,3,3,3
    facearray[28] = 4,3,4,1,4,2,4,4,4,0
    D3DCOMMAND pyr,@ADDFACES,facearray

    'finally initialize the custom shape
    D3DCOMMAND pyr,@CUSTOMINIT
RETURN pyr
```

The above code creates a pyramid shape on a DirecX 3D Screen, and allows you to use the arrow keys to move around the shape.

See Also

CREATESCREEN

CREATEDIR

Usage

```
CREATEDIR newdirname
```

Parameters

`newdirname` a string containing the full path to the directory. This should not end with a "\".

Returns

Returns 0 on error.

Description

Creates a new directory as specified by newdirname.

Examples

```
OPENCONSOLE
DEF returnVal : INT
PRINT "Creating Directory 'C:\Creative BASICTest'"
returnVal = CREATEDIR("C:\Creative BASICTest")
IF returnVal = 0
    PRINT "An error occurred.  The specified directory could not be created!"
ELSE
    PRINT "Directory 'C:\Creative BASICTest' was created successfully!"
ENDIF
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program will create a new directory on your C drive called Creative BASICTest. If you run this program more than once, it will show an error. This is because the directory already exists.

See Also

REMOVEDIR

CREATESCREEN

Usage

```
CREATESCREEN winhandle,width,height {,bpp}
```

Parameters

winhandle	user defined WINDOW variable.
width	the width (in pixels) of the screen you want to create.
height	the height (in pixels) of the screen you want to create.
bpp	Bits Per Pixel. The color depth of the screen (i.e. 8bit, 16bit, 32bit). If you include this, a fullscreen window will open using the color depth specified. If you do not include this, a window of the specified width and height will be opened. This window will use the color settings currently in use by the PC it's running on.

Returns

Returns a value greater than or equal to 0.

Description

Creates a DirectX screen and attaches it to the window specified. If bpp is specified the DirectX screen will be full screen. Otherwise it will be a windowed surface.

Examples

```
DEF myWindow:WINDOW
DEF state:INT

WINDOW myWindow,0,0,640,480,@NOAUTODRAW,0,"Create Screen",mainwindow
IF CREATESCREEN(myWindow,640,480) < 0
    MESSAGEBOX myWindow, "Could not create DirectX screen","Error"
    CLOSEWINDOW myWindow
    END
ENDIF

DXFILL myWindow, RGB(0,0,255)
```

```
DXFLIP myWindow
state = 0

STARTTIMER myWindow,500

run = 1

WAITUNTIL run = 0
STOPTIMER myWindow
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
    CASE @IDDXUPDATE
        IF state THEN DXFILL(myWindow,RGB(0,0,255)) ELSE DXFILL(myWindow,RGB(0,255,0))
        DXFLIP myWindow
    CASE @IDTIMER
        state = NOT state
ENDSELECT
RETURN
```

The above example opens a DirectX window and changes the fill color of the window based on a timer.

See Also

DXFILL; DXFLIP; CREATE3DSCREEN

D3DCAMERA()

Usage

```
D3DCAMERA(scene)
```

Parameters

`scene` int reference to a scene.

Returns

`camera` int reference to a camera.

Description

Create a camera frame and attaches it to the provided scene. Camera frames create the 'view' of the 3D world. You can have as many camera frames as needed to support multiple views. Returns an INT value that can be used as a parameter to other commands.

Examples

```
DEF myWindow:window
DEF scene,camera,shape,light:int

WINDOW myWindow,0,0,640,480,@CAPTION|@NOAUTODRAW,0,"D3D",main
CREATE3DSCREEN(myWindow,640,480)
D3DSETQUALITY myWindow,@LIGHTON | @FILLSOLID | @SHADEGOURAUD

scene = D3DSCENE(myWindow)
```

```
camera = D3DCAMERA(scene)

light = D3DLIGHT(scene,@LIGHTDIRECTIONAL,1.4,1.4,1.4)
D3DCOMMAND light,@SETPOSITION,scene,25,-25,0
D3DCOMMAND light,@SETORIENTATION,scene,-1,0,1,0,1,0

shape = D3DSHAPE(scene,@SHAPESPHERE,10,30)
D3DCOMMAND shape,@SETPOSITION,scene,0,0,50

SETWINDOWCOLOR myWindow,RGB(0,0,0)
DXFILL myWindow,RGB(0,0,0)

run = 1
waituntil run=0

D3DDELETE light
D3DDELETE shape
D3DDELETE camera
D3DDELETE scene
CLOSEWINDOW myWindow
END

SUB main
    SELECT @class
        CASE @IDCLOSEWINDOW
            run=0
        CASE @IDDXUPDATE
            D3DRENDER scene,camera
            DXFLIP myWindow,0,0
    ENDSELECT
RETURN
```

A window is opened, and a Direct3D screen is attached to it. D3DSCENE is then used to create the master scene frame, and a camera is created (using D3DCAMERA) within the scene. The rest of the program creates a lightsource and a sphere, with the *main* subroutine handling the drawing of the 3D scene.

See Also

D3DSCENE; D3DRENDER; DXFLIP; D3DDELETE

D3DCOMMAND()

Usage

```
D3DCOMMAND(frame, option {,parameters})
```

Parameters

frame	a frame in the 3d scene.
option	the option to apply to the frame.
parameters	optional, parameters to pass to the command.

Returns

Nothing.

Description

Using D3DCOMMAND enables you to modify a variety of parameter for any of the frames within your scene. For details of the various options, please see the D3DCommand Reference.

Examples

```
DEF win:window
DEF scene,camera,shape,light:int

WINDOW win,0,0,640,480,@CAPTION|@NOAUTODRAW,0,"D3D",main
CREATE3DSCREEN(win,640,480)
D3DSETQUALITY win,@LIGHTON | @FILLSOLID | @SHADEGOURAUD

scene = D3DSCENE(win)
camera = D3DCAMERA(scene)

light = D3DLIGHT(scene,@LIGHTDIRECTIONAL,1.4,1.4,1.4)
D3DCOMMAND light,@SETPOSITION,scene,25,-25,0
D3DCOMMAND light,@SETORIENTATION,scene,-1,0,1,0,1,0

shape = D3DSHAPE(scene,@SHAPESPHERE,10,30)
D3DCOMMAND shape,@SETPOSITION,scene,0,0,50

SETWINDOWCOLOR win,RGB(0,0,0)
DXFILL win,RGB(0,0,0)

run = 1
waituntil run=0

D3DDELETE light
D3DDELETE shape
D3DDELETE camera
D3DDELETE scene
CLOSEWINDOW win
END

SUB main
    SELECT @class
        CASE @IDCLOSEWINDOW
            run=0
        CASE @IDDXUPDATE
            D3DRENDER scene,camera
            DXFLIP win,0,0
    ENDSELECT
RETURN
```

In the example, D3DCOMMAND is used with the *@SETPOSITION* option to place the light and the

sphere within the scene, and with the *@SETORIENTATION* option to adjust the directon of the light source.

D3DDELETE

Usage

```
D3DDELETE frame
```

Parameters

frame an INT reference to 3D frame.

Returns

Nothing.

Description

Deletes a previously created scene, camera, light or shape frame. All frames must be deleted before your program exits or the system will lose the memory allocated by those frames. The frame is removed from the hierarchy before being deleted. This allows removing 3D elements while the program is running.

Examples

```
DEF myWindow:WINDOW
DEF scene,run:INT

WINDOW myWindow,0,0,640,480,@CAPTION|@NOAUTODRAW,0,"D3D Delete",main
CREATE3DSCREEN(myWindow,640,480)

scene = D3DSCENE(myWindow)
'CLEAR THE VIDEO MEMORY
DXFILL myWindow,RGB(0,0,0)
run = 1
WAITUNTIL run = 0

D3DDELETE scene
CLOSEWINDOW myWindow
END

SUB main
    SELECT @class
        CASE @IDCLOSEWINDOW
            run = 0
        CASE @IDDXUPDATE
            DXFLIP myWindow
    ENDSELECT
RETURN
```

This creates a D3D scene, and then waits for the window to be closed. When the window close button is clicked, the D3DSCENE command is used to delete the created scene frame, and then the window is closed.

See Also

D3DSCENE; D3DCAMERA; D3DRENDER; DXFLIP

D3DLIGHT()

Usage

```
D3DLIGHT(parent, type, R, G, B)
```

Parameters

parent	the parent frame of the light.
type	the type of light: @LIGHTAMBIENT @LIGHTPOINT @LIGHTSPOT @LIGHTDIRECTIONAL @LIGHTPARALLELPOINT
R	the amount of red in the light source.
G	the amount of green in the light source.
B	the amount of blue in the light source.

Returns

An INT handle.

Description

Creates a light source in the parent frame specified. R, G, B are floating point values that specify the color and intensity of the light.

Examples

```
DEF win:window
DEF scene,camera,shape,light:int

WINDOW win,0,0,640,480,@CAPTION|@NOAUTODRAW,0,"D3D",main
CREATE3DSCREEN(win,640,480)
D3DSETQUALITY win,@LIGHTON | @FILLSOLID | @SHADEGOURAUD

scene = D3DSCENE(win)
camera = D3DCAMERA(scene)

light = D3DLIGHT(scene,@LIGHTDIRECTIONAL,1.4,1.4,1.4)
D3DCOMMAND light,@SETPOSITION,scene,25,-25,0
D3DCOMMAND light,@SETORIENTATION,scene,-1,0,1,0,1,0

shape = D3DSHAPE(scene,@SHAPESPHERE,10,30)
D3DCOMMAND shape,@SETPOSITION,scene,0,0,50

SETWINDOWCOLOR win,RGB(0,0,0)
DXFILL win,RGB(0,0,0)

run = 1
waituntil run=0

D3DDELETE light
D3DDELETE shape
D3DDELETE camera
D3DDELETE scene
CLOSEWINDOW win
END
```

```
SUB main
    SELECT @class
        CASE @IDCLOSEWINDOW
            run=0
        CASE @IDDXUPDATE
            D3DRENDER scene,camera
            DXFLIP win,0,0
    ENDSELECT
RETURN
```

In the example, D3DLIGHT is used to create a directional light source.

See Also

D3DCOMMAND; D3DSCENE; D3DCAMERA

D3DMOVE

Usage

```
D3DMOVE winhandle, amount
```

Parameters

winhandle	a user defined WINDOW variable.
amount	a numeric value.

Returns

Nothing.

Description

Moves all frames in the the window using their current directional velocity, rotational and transformation vectors. Movement is multiplied by the amount specified. By keeping track of display rates you can adjust the amount parameter to keep the perception of speed constant on all systems. The normal amount would be 1, with higher values resulting in faster movements, and lower values slowing the overall movement.

Examples

```
DEF myWindow:window
DEF scene,camera,shape,light:int

WINDOW myWindow,0,0,640,480,@CAPTION|@NOAUTODRAW,0,"D3D",main
CREATE3DSCREEN(myWindow,640,480)
D3DSETQUALITY myWindow,@LIGHTON | @FILLSOLID | @SHADEGOURAUD

scene = D3DSCENE(myWindow)
camera = D3DCAMERA(scene)

light = D3DLIGHT(scene,@LIGHTDIRECTIONAL,1.4,1.4,1.4)
D3DCOMMAND light,@SETPOSITION,scene,25,-25,0
D3DCOMMAND light,@SETORIENTATION,scene,-1,0,1,0,1,0

shape = D3DSHAPE(scene,@SHAPESPHERE,10,30)
D3DCOMMAND shape,@SETPOSITION,scene,0,0,50
D3DCOMMAND shape, @SETVELOCITY, scene, 0.1, 0, 0, 0

SETWINDOWCOLOR myWindow,RGB(0,0,0)
```

```
DXFILL myWindow,RGB(0,0,0)

run = 1
WAITUNTIL run=0

D3DDELETE light
D3DDELETE shape
D3DDELETE camera
D3DDELETE scene
CLOSEWINDOW myWindow
END

SUB main
    SELECT @class
        CASE @IDCLOSEWINDOW
            run = 0
        CASE @IDDXUPDATE
            D3DMOVE myWindow, 1
            D3DRENDER scene,camera
            DXFLIP myWindow,0,0
    ENDSELECT
RETURN
```

This create a simple 3D scene with a sphere. The sphere is given a velocity of 0.1 (using `D3DCOMMAND` with the *@SETVELOCITY* parameter) and is therefore moved with each call of the `D3DMOVE` command in the *main* subroutine.

See Also

`D3DCOMMAND`; `D3DSHAPE`

D3DRENDER

Usage

```
D3DRENDER scene, camera
```

Parameters

`scene`	a handle to a scene.
`camera`	a handle to a camera.

Returns

Nothing.

Description

Draws all of the objects in the specified scene from the viewpoint of the given camera.

Examples

```
DEF win:window
DEF scene,camera,shape,light:int

WINDOW win,0,0,640,480,@CAPTION|@NOAUTODRAW,0,"D3D",main
CREATE3DSCREEN(win,640,480)
D3DSETQUALITY win,@LIGHTON | @FILLSOLID | @SHADEGOURAUD

scene = D3DSCENE(win)
```

```
camera = D3DCAMERA(scene)

light = D3DLIGHT(scene,@LIGHTDIRECTIONAL,1.4,1.4,1.4)
D3DCOMMAND light,@SETPOSITION,scene,25,-25,0
D3DCOMMAND light,@SETORIENTATION,scene,-1,0,1,0,1,0

shape = D3DSHAPE(scene,@SHAPESPHERE,10,30)
D3DCOMMAND shape,@SETPOSITION,scene,0,0,50

SETWINDOWCOLOR win,RGB(0,0,0)
DXFILL win,RGB(0,0,0)

run = 1
WAITUNTIL run=0

D3DDELETE light
D3DDELETE shape
D3DDELETE camera
D3DDELETE scene
CLOSEWINDOW win
END

SUB main
    SELECT @class
        CASE @IDCLOSEWINDOW
            run=0
        CASE @IDDXUPDATE
            D3DRENDER scene,camera
            DXFLIP win,0,0
    ENDSELECT
RETURN
```

This creates a simple 3D scene. The graphics aren't actually rendered until the `D3DRENDER` command is called, and then are only displayed once the `DXFLIP` command is executed.

See Also

DXFLIP

D3DSCENE()

Usage

```
D3DSCENE(window)
```

Parameters

`winhandle` a D3D window.

Returns

Nothing

Description

Creates a scene for the specified 3D window. This is the top level of the 3D hierarchy.

Examples

```
DEF myWindow:window
```

```
DEF scene,camera,shape,light:int

WINDOW myWindow,0,0,640,480,@CAPTION|@NOAUTODRAW,0,"D3D",main
CREATE3DSCREEN(myWindow,640,480)
D3DSETQUALITY myWindow,@LIGHTON | @FILLSOLID | @SHADEGOURAUD

scene = D3DSCENE(myWindow)
camera = D3DCAMERA(scene)

light = D3DLIGHT(scene,@LIGHTDIRECTIONAL,1.4,1.4,1.4)
D3DCOMMAND light,@SETPOSITION,scene,25,-25,0
D3DCOMMAND light,@SETORIENTATION,scene,-1,0,1,0,1,0

shape = D3DSHAPE(scene,@SHAPESPHERE,10,30)
D3DCOMMAND shape,@SETPOSITION,scene,0,0,50

SETWINDOWCOLOR myWindow,RGB(0,0,0)
DXFILL myWindow,RGB(0,0,0)

run = 1
WAITUNTIL run=0

D3DDELETE light
D3DDELETE shape
D3DDELETE camera
D3DDELETE scene
CLOSEWINDOW win
END

SUB main
    SELECT @class
        CASE @IDCLOSEWINDOW
            run=0
        CASE @IDDXUPDATE
            D3DRENDER scene,camera
            DXFLIP myWindow,0,0
    ENDSELECT
RETURN
```

As seen in the example, the first thing that you must do when using the 3D commands is to create a scene. Most of the other 3D commands use this scene as a reference (for positioning etc).

See Also

D3DRENDER; DXFLIP; D3DDELETE

D3DSETQUALITY

Usage

```
D3DSETQUALITY winhandle, option { | option | option...}
```

Parameters

`winhandle` a window containing 3d content.

`option` a combination of the following as required:

@LIGHTON	enable lighting.
@LIGHTOFF	disable lighting.
@FILLPOINTS	the image is drawn using a single point at the vertex of each polygon.
@FILLWIREFRAME	a wireframe image is rendered.
@FILLSOLID	solid 3d objects are rendered.
@SHADEFLAT	flat shading is used.
@SHADEGOURAUD	gouraud shading is used - this produces better quality (smoother) visual output.

Returns

Nothing

Description

This sets the default rendering quality of the 3d engine.

Examples

```
DEF myWindow:window
DEF scene,camera,shape,light:int

WINDOW myWindow,0,0,640,480,@CAPTION|@NOAUTODRAW,0,"D3D",main
CREATE3DSCREEN(myWindow,640,480)
D3DSETQUALITY myWindow,@LIGHTON | @FILLWIREFRAME | @SHADEGOURAUD

scene = D3DSCENE(myWindow)
camera = D3DCAMERA(scene)

light = D3DLIGHT(scene,@LIGHTDIRECTIONAL,1.4,1.4,1.4)
D3DCOMMAND light,@SETPOSITION,scene,25,-25,0
D3DCOMMAND light,@SETORIENTATION,scene,-1,0,1,0,1,0

shape = D3DSHAPE(scene,@SHAPESPHERE,10,30)
D3DCOMMAND shape,@SETPOSITION,scene,0,0,50

SETWINDOWCOLOR myWindow,RGB(0,0,0)
DXFILL myWindow,RGB(0,0,0)

run = 1
WAITUNTIL run=0

D3DDELETE light
D3DDELETE shape
D3DDELETE camera
D3DDELETE scene
CLOSEWINDOW myWindow
END
```

```
SUB main
    SELECT @class
        CASE @IDCLOSEWINDOW
            run=0
        CASE @IDDXUPDATE
            D3DRENDER scene,camera
            DXFLIP myWindow,0,0
    ENDSELECT
RETURN
```

Creates a simple 3D scene using wireframe rendering, this option being set using the D3DSETQUALITY command.

D3DSETRENDERMODE

Usage

```
D3DSETRENDERMODE window, mode
```

Parameters

winhandle	a user defined WINDOW variable.
mode	a combination of the following @RMBLENDED @RMSTIPPLED @RMSORTED @RMDISABLESORTED @RMDEPENDENTSPECULAR

Returns

Nothing.

Description

Sets the alpha blending (opacity) and object sorting mode for the 3D engine.

Examples

```
DEF win:window
DEF scene,camera,shape,shape2,light:int

WINDOW win,0,0,640,480,@CAPTION|@NOAUTODRAW,0,"D3D",main
CREATE3DSCREEN(win,640,480)
D3DSETQUALITY win,@LIGHTON | @FILLSOLID | @SHADEGOURAUD
D3DSETRENDERMODE win,@RMBLENDED | @RMSORTED

scene = D3DSCENE(win)
camera = D3DCAMERA(scene)

light = D3DLIGHT(scene,@LIGHTDIRECTIONAL,1.4,1.4,1.4)
D3DCOMMAND light,@SETPOSITION,scene,25,-25,0
D3DCOMMAND light,@SETORIENTATION,scene,-1,0,1,0,1,0

shape = D3DSHAPE(scene,@SHAPESPHERE,20,30)
D3DCOMMAND shape,@SETPOSITION,scene,0,0,50
```

```
D3DCOMMAND shape,@SETSHAPECOLOR, 1,1,0,0.4

shape2 = D3DSHAPE(scene,@SHAPESPHERE,10,15)
D3DCOMMAND shape2,@SETPOSITION,scene,0,0,50

run = 1
waituntil run=0

D3DDELETE light
D3DDELETE shape
D3DDELETE camera
D3DDELETE scene
CLOSEWINDOW win
END

SUB main
    SELECT @class
        CASE @IDCLOSEWINDOW
            run=0
        CASE @IDDXUPDATE
            D3DRENDER scene,camera
            DXFLIP win,0,0
    ENDSELECT
RETURN
```

A simple 3d scene is opened, and a semi-transparent sphere is created using D3DCOMMAND with the @SETSHAPECOLOR option. A second sphere is placed within the first. The transparency mode is set to alpha blending with buffered transparent polygons (the D3DSETRENDERMODE with the @RMSORTED option). This makes the second sphere visable inside the first.

See Also

D3DSETQUALITY, D3DCOMMAND

D3DSHAPE()

Usage

```
D3DSHAPE(parent ,type {,parameters})
```

Parameters

parent	an INT reference to a parent frame.
type	the type of shape to create: @SHAPECUSTOM @SHAPESPHERE @SHAPECUBE
parameters	optional. Dependant on the type of object required.

Returns

An inter value which references the created shape.

Description

Creates either a blank shape that the user can then load a custom shape in to, a sphere primitive, or a cube primitive.

Examples

```
DEF win:window
DEF scene,camera,shape,light:int

WINDOW win,0,0,640,480,@CAPTION|@NOAUTODRAW,0,"D3D",main
CREATE3DSCREEN(win,640,480)
D3DSETQUALITY win,@LIGHTON | @FILLSOLID | @SHADEGOURAUD

scene = D3DSCENE(win)
camera = D3DCAMERA(scene)

light = D3DLIGHT(scene,@LIGHTDIRECTIONAL,1.4,1.4,1.4)
D3DCOMMAND light,@SETPOSITION,scene,25,-25,0
D3DCOMMAND light,@SETORIENTATION,scene,-1,0,1,0,1,0

shape = D3DSHAPE(scene,@SHAPESPHERE,10,30)
D3DCOMMAND shape,@SETPOSITION,scene,0,0,50

SETWINDOWCOLOR win,RGB(0,0,0)
DXFILL win,RGB(0,0,0)

run = 1
waituntil run=0

D3DDELETE light
D3DDELETE shape
D3DDELETE camera
D3DDELETE scene
CLOSEWINDOW win
END

SUB main
    SELECT @class
        CASE @IDCLOSEWINDOW
            run=0
        CASE @IDDXUPDATE
            D3DRENDER scene,camera
            DXFLIP win,0,0
    ENDSELECT
RETURN
```

Creates a simple 3d scene. The sphere is created using D3DSHAPE, and is assigned to the vairable shape.

See Also

D3DCOMMAND

DATE$

Usage

```
DATE$ dateformat
```

Parameters

dateformat — optional. A string describing how you want the date formatted. Can use any combination of the following

d	Day of month as digits with no leading zero for single-digit days.
dd	Day of month as digits with leading zero for single-digit days.
ddd	Day of week as a three-letter abbreviation..
dddd	Day of week as its full name.
M	Month as digits with no leading zero for single-digit months.
MM	Month as digits with leading zero for single-digit months.
MMM	Month as a three-letter abbreviation.
MMMM	Month as its full name.
y	Year as last two digits, but with no leading zero for years less than 10.
yy	Year as last two digits, but with leading zero for years less than 10.
yyyy	Year represented by full four digits.

to add text in the ouput use the single quote so as not to confuse with formatting characters.

Returns

Returns the current date as a string.

Description

Date has the format MM-DD-YYYY

Examples

```
OPENCONSOLE
PRINT "Today's date is : ", DATE$
PRINT
PRINT DATE$ "'Todays date is'  ddd, dd MMM yyyy"
PRINT "Press any key to end this program"
DO
UNTIL INKEY$ <>""
CLOSECONSOLE
END
```

The above program will print out a piece of text, displaying the current date. Notice how the literal string "Todays date is" is surrounded by ' single quotes. While the DATE$ command will format any string passed to it, it will try and replace d, M, y characters with the date information.

See Also

TIME$()

DECLARE

Usage

```
DECLARE sub-name({parameters})

DECLARE "{!}dll-name", name({parameter,...}){,return}

DECLARE "{!}dll-name", local ALIAS name({parameter,...}){,return}
```

Parameters

subname — name of the subroutine to declare or DLL.

parameters — optional, if the subroutine has parameters they are declared here.

Returns

Nothing

Description

Declares a user subroutine or DLL call. Optional parameter list and return type. Return type is only used for DLL calls. Optional ! symbol specifies the CDECL calling convention.

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Demo",main
DECLARE "User32",MessageBoxA (wnd:window,text:string,title:string,flags:int),int
MessageBoxA (myWindow,"This messagebox wasn't activated using CALL","Information",0)

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

Declares a subroutine named MySubroutine which accepts an int called row and a string called text$ as input. Also declares a DLL, User32 with the MessageBoxA parameter.

See Also

ALIAS; SUB

DEFAULT

Usage

```
 DEFAULT
```

Parameters

None

Returns

Nothing

Description

Defines a `DEFAULT` execution point if all the `CASE` statements in a `SELECT` statement are `FALSE`.

Examples

```
A = 5
SELECT A
     CASE 1
           PRINT "1!"
     CASE 2
           PRINT "2"
     DEFAULT
           PRINT "None of the above"
ENDSELECT
```

In the above select case, the variable A is checked against 1 and 2, in this example it is 5 so neither are true and the default block is executed.

See Also
SELECT; CASE

DELETE

Usage
```
DELETE pointer
```
Parameters
`pointer` a variable created with the `NEW` function.
Returns
Nothing
Description
Deletes a dynamically created variable. Pointer must point to a variable created with the `NEW` function.
Examples
```
OPENCONSOLE
DEF p : POINTER

'Create an INT in memory
p = NEW(INT, 1)
#p = 75
PRINT #p

'delete the INT
DELETE p

PRINT "Press any key to close"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

This defines `p` as a pointer, and creates a new `INT` variable, assigning it to `p`. The value of 75 if placed in the address pointed to by `p` (using the # operator), and printed to the screen. The memory used by the variable is then freed using the `DELETE` command.
See Also
NEW

DELETEFILE

Usage
```
DELETEFILE file
```
Parameters
`file` a fully qualified path to a file.
Returns
0 on error.
Description
Deletes the file specified. `DELETEFILE` returns 0 if the file could not be deleted.
Examples
```
deletefile("c:\test.txt")
```

WARNING, this command deletes files.

The example would delete a file called text.txt from the root of C drive, point it to a file which you wish to delete.

DELETEIMAGE

Usage

```
DELETEIMAGE imagehandler, type
```

Parameters

imagehandler	an image.
type	type of image. **@IMGBITMAP** Loads a bitmap from a file or resource **@IMGICON** Loads an icon from a file or resource **@IMGCURSOR** Loads a cursor from a file or resource **@IMGEMF** Loads an enhanced meta file (EMF) from a disk file **@IMGSCALABLE** Loads a scalable bitmap, JPG, GIF or TIFF image from a file or resource **@IMGOEM** OR in with @IMGBITMAP, @IMGICON or @IMGCURSOR to load an OEM (system) image. **@IMGMAPCOLORS** OR in with @IMGBITMAP to have the system search the bitmap color table and map shades of gray to standard system 3D colors.

Returns

nothing

Description

Frees memory used by an image previously loaded with the LOADIMAGE statement. TYPE must be the same as specified in the LOADIMAGE statement.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT
DEF pic AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "DELETEIMAGE Example",
main

pic = LOADIMAGE("C:\mypic.bmp",@IMGBITMAP)
SHOWIMAGE myWindow,pic,@IMGBITMAP,0,0,100,100

run = 1
WAITUNTIL run = 0
DELETEIMAGE pic,@IMGBITMAP
END

SUB main
   IF @CLASS = @IDCLOSEWINDOW
      CLOSEWINDOW myWindow
      run = 0
   ENDIF
RETURN
```

An image is loaded and given the handle "pic". The image is then displayed in the window using the SHOWIMAGE command. Once the window is closed the image is deleted with DELETEIMAGE.

See Also

LOADIMAGE

DELETESTRING

Usage

```
DELETESTRING winhandle,controlID,position
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
controlID	the ID number of the listbox or combo box.
position	the position of the option you want to delete.

Returns

Nothing

Description

Removes a string from a list box or combo box control. Remaining strings are moved up to fill the empty position.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX | @HSCROLL | @VSCROLL, 0,
"DeleteString Example", main

CONTROL myWindow,"L,ListBox1,115,29,72,121,0x50800140,1"
CONTROL myWindow,"B,Delete,20,72,70,20,0x50000000,2"

ADDSTRING myWindow,1,"1"
ADDSTRING myWindow,1,"2"
ADDSTRING myWindow,1,"3"

SETSELECTED myWindow,1,1

run = 1
WAITUNTIL run = 0

END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
IF (@CLASS = @IDCONTROL)
  DELETESTRING myWindow, 1, 0
ENDIF
RETURN
```

Deletes strings from position 0 in the combo box each time the button is pressed.

See Also

ADDSTRING

DIALOG

Usage

```
DIALOG name, left, top, width, height, flags, parent, title, handler
```

Parameters

name	variable name of the DIALOG variable used to store the dialog.
left	left edge of the dialog.
top	top edge of the dialog.
width	width of the dialog.
height	height of the dialog.
flags	numeric value, specifies style creation flags.
parent	WINDOW variable of parent window if this dialog is a child, otherwise 0.
title	the text to be shown in the caption of the dialog.
handler	the name of a subroutine to handle messages for the dialog.

Returns

nothing

Description

Defines a dialog to be used with the DOMODAL or SHOWDIALOG statements. The name must first be defined using the DEF statement as type DIALOG.

Examples

```
DEF d1:DIALOG
DEF w:WINDOW
DEF result:INT
DEF answer:STRING
'Open our window and define a dialog
WINDOW w,0,0,640,200,@SIZE,0,"Dialog Test",wndproc
DIALOG d1,0,0,100,100,@CAPTION|@SYSMENU,w,"My Dialog",dialoghandler
CONTROL d1,"B,OK,25,75,50,20,@TABSTOP,1"
CONTROL d1,"E,,15,45,70,14,@TABSTOP,2"
'Show the dialog
result = DOMODAL d1
'Print the result
MOVE w,4,20
IF result = @IDOK
        PRINT w, answer
ELSE
        PRINT w, "DIALOG canceled"
ENDIF
'Just wait for the window to be closed
run=1
WAITUNTIL run = 0

CLOSEWINDOW w
END

'Our window subroutine
```

```
SUB wndproc
SELECT @CLASS
   CASE @IDCLOSEWINDOW
    run = 0
ENDSELECT
RETURN

'Our dialog subroutine
SUB dialoghandler
SELECT @CLASS
   CASE @IDCONTROL
        SELECT @CONTROLID
           CASE 1
                answer = GETCONTROLTEXT(d1, 2)
                CLOSEDIALOG d1,@IDOK
        ENDSELECT
'All controls should be initialized while processing the @IDINITDIALOG message
   CASE @IDINITDIALOG
       CENTERWINDOW d1
        SETCONTROLTEXT d1,2,"Yipee!"
ENDSELECT
RETURN
```

Defines a `DIALOG` called `myDialog`, which is 100 x 100 to open at 10,10.

See Also

SHOWDIALOG;DOMODAL;WINDOW;CLOSEDIALOG

DEF

Usage

```
 DEF name{[x{,y{,z}}]{,name...}:type

-----OR-----


DEF name{[x{,y{,z}}]{,name...} AS type
```

Parameters

`name`	the name you wish to call your variable.
`x,y,z`	optional, other variables of the same type.
`type`	the type of variable.

Returns

Nothing

Description

Defines a variable to a predefined or user defined type. Define an array of up to 3 dimensions by following the name with brackets and dimension amounts.

Examples

```
DEF A$,B$:STRING
DEF num:FLOAT
DEF num2[20] AS DOUBLE
```

Defines A$ and B$ as strings, num as a float and num2[20] as an array from (0-19) of doubles.

DO

Usage

```
 DO

      code to be looped

UNTIL conditon
```

Parameters

None

Returns

Nothing

Description

Begins a DO UNTIL loop. see UNTIL for more information

Examples

```
OPENCONSOLE
DEF x:INT
x=1

DO
    PRINT x
    x=x+1
UNTIL x > 10

PRINT "Press Any Key To Close"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Prints and then increments the variable x until x is greater than 10.

See Also

UNTIL

DOMODAL()

Usage

```
 DOMODAL( dialog )
```

Parameters

dialog a user defined DIALOG variable.

Returns

Nothing

Description

Opens a dialog previously declared with a DIALOG statement.

Examples

```
DEF d1:DIALOG
DEF w:WINDOW
DEF result:INT
DEF answer:STRING
```

```
'Open our window and define a dialog
WINDOW w,0,0,640,200,@SIZE,0,"Dialog Test",wndproc
DIALOG d1,0,0,100,100,@CAPTION|@SYSMENU,w,"My Dialog",dialoghandler
CONTROL d1,"B,OK,25,75,50,20,@TABSTOP,1"
CONTROL d1,"E,,15,45,70,14,@TABSTOP,2"
'Show the dialog
result = DOMODAL d1
'Print the result
MOVE w,4,20
IF result = @IDOK
        PRINT w, answer
ELSE
        PRINT w, "DIALOG canceled"
ENDIF
'Just wait for the window to be closed
run=1
WAITUNTIL run = 0

CLOSEWINDOW w
END

'Our window subroutine
SUB wndproc
SELECT @CLASS
   CASE @IDCLOSEWINDOW
    run = 0
ENDSELECT
RETURN

'Our dialog subroutine
SUB dialoghandler
SELECT @CLASS
   CASE @IDCONTROL
        SELECT @CONTROLID
           CASE 1
                answer = GETCONTROLTEXT(d1, 2)
                CLOSEDIALOG d1,@IDOK
        ENDSELECT
'All controls should be initialized while processing the @IDINITDIALOG message
   CASE @IDINITDIALOG
       CENTERWINDOW d1
        SETCONTROLTEXT d1,2,"Yipee!"
ENDSELECT
RETURN
```

Opens up a modal dialog window once the button is pressed.

See Also

DIALOG;SHOWDIALOG

DRAWMODE

Usage

```
DRAWMODE winhandle,mode
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
	the mode to use.
mode	**@OPAQUE** Background is filled with the current background color before the text or line is drawn. This is the default background mode. **@TRANSPARENT** Background is not changed before drawing

Returns

nothing

Description

Sets the background fill mode. The background mode defines whether the system removes existing background colors on the drawing surface before drawing text or any non-solid line. Mode is either @TRANSPARENT or @OPAQUE

Examples

```
DEF w1:WINDOW
WINDOW w1,0,0,640,480,@NOAUTODRAW|@SIZE,0,"DrawMode",mainwindow

CREATESCREEN(w1,640,480)

run = 1
WAITUNTIL run = 0
CLOSEWINDOW w1
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN

SUB update
    'clear screen
    DXFILL w1,RGB(0,0,0)

    FRONTPEN w1, RGB(0,0,255)
    DRAWMODE w1,@OPAQUE
```

```
    MOVE w1,0,0
    PRINT w1,"OPAQUE!"
    DRAWMODE w1,@TRANSPARENT
    MOVE w1,0,30
    PRINT w1,"TRANSPARENT!"


    'show the changes
    DXFLIP w1
RETURN
```

Prints two lines of text to the screen, one with transparency draw mode and the other with opaque.

DXCREATEMAP

Usage

```
DXCREATEMAP winhandle, width, height, fill {,scrollwrap}
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
width	map tile width.
height	map tile height.
fill	fill tile.
scrollwrap	optional, if map should be scroll wrapped.

Returns

0 on failure

Description

Creates a map of width and height tiles filled with the tile number specified. Optional scrollwrap = 1 to wrap the images when the map is moved past the boundaries of the screen. Returns 0 on failure.

Examples

```
DEF myWindow:WINDOW
DEF state1,state2,spr1,spr2,x,y1,y2,bitmap,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@NOCAPTION,0,"",mainwindow
CREATESCREEN(myWindow,640,480,16)
SETFONT myWindow,"Courier New",12,700
DRAWMODE myWindow,@TRANSPARENT
FRONTPEN myWindow,RGB(255,255,0)

'load the tile images for the map
DXNEWMAP(myWindow,"C:\tiles.bmp",64,64,0)
'create a blank map filled with tile#1 and scroll wrapped
DXCREATEMAP myWindow,10,10,1,1

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
```

```
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
CASE @IDCHAR
    SELECT @code
        CASE ASC("Q")
        CASE ASC("q")
            run = 0
    ENDSELECT
ENDSELECT
RETURN

SUB update
    'check for arrow keys and scroll appropriately
    if GETKEYSTATE(37) THEN DXSCROLLMAP(myWindow,@SCROLLLEFT,2)
    if GETKEYSTATE(38) THEN DXSCROLLMAP(myWindow,@SCROLLUP,2)
    if GETKEYSTATE(39) THEN DXSCROLLMAP(myWindow,@SCROLLRIGHT,2)
    if GETKEYSTATE(40) THEN DXSCROLLMAP(myWindow,@SCROLLDOWN,2)
    'draw the map to the back buffer
    DXDRAWMAP myWindow
    'draw a message
    MOVE myWindow,10,10
    PRINT myWindow,"Map Example - Press Q to quit "
    MOVE myWindow,10,25
    PRINT myWindow,"Use arrow keys to scroll around"
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads in tiles with DXNEWMAP then creates a blank map with DXCREATEMAP, and fills it with the tile. Tiles are loaded from C:\Tiles.bmp you can find the Tiles.bmp in your Creative BASIC samples folder, change this path to point to it.

See Also

DXLOADMAP;DXNEWMAP

DXDRAWALLSPRITES

Usage

```
DXDRAWALLSPRITES winHandle
```

Parameters

winHandle a user defined WINDOW.

Returns

Nothing.

Description

Draws all sprites contained in the window to the back buffer of the screen.

Examples

```
DEF myWindow:WINDOW
DEF spr1,spr2,spr3:INT
```

```
WINDOW myWindow, 0, 0, 640, 480, @NOAUTODRAW|@SIZE, 0, "DXSprite", main
CREATESCREEN(myWindow,640,480)

'load the sprite
spr1 =  DXSPRITE(myWindow, "C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp", 40, 36, 1, RGB(87,87,87))
spr2 =  DXSPRITE(myWindow, "C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp", 40, 36, 2, RGB(87,87,87))
spr3 =  DXSPRITE(myWindow, "C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp", 40, 36, 3, RGB(87,87,87))

DXMOVESPRITE spr1, 100, 100
DXMOVESPRITE spr2, 200, 200
DXMOVESPRITE spr3, 300, 300

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

main:
    SELECT @CLASS
        CASE @IDDXUPDATE
            GOSUB update
        CASE @IDCLOSEWINDOW
            run = 0
    ENDSELECT
RETURN

SUB update
    'clear screen
    DXFILL myWindow,RGB(0,0,0)
    'draw sprite
    DXDRAWALLSPRITES myWindow
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads 3 sprites from the file `C:\Program Files\Ionic Wind\Creative BASIC\samples\mouth.bmp` (change as needed) and then renders all the sprites to the screen with DXDRAWALLSPRITES.

See Also

DXSPRITE; DXDRAWSPRITE

DXDRAWALLSPRITES

Usage

```
DXDRAWALLSPRITES window
```

Parameters

`window` a window

Returns

nothing

Description

Draws all sprites contained in the window to the back buffer of the screen.

Examples

```
DEF w1:WINDOW
DEF spr1,spr2,spr3:INT
WINDOW w1,0,0,640,480,@NOAUTODRAW|@SIZE,0,"DXSprite",mainwindow

CREATESCREEN(w1,640,480)

'load the sprite
spr1 =  DXSPRITE(w1,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,1,RGB(87,87,87))
spr2 =  DXSPRITE(w1,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,2,RGB(87,87,87))
spr3 =  DXSPRITE(w1,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,3,RGB(87,87,87))

DXMOVESPRITE spr1,100,100
DXMOVESPRITE spr2,200,200
DXMOVESPRITE spr3,300,300

run = 1
WAITUNTIL run = 0
CLOSEWINDOW w1
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
                GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN

SUB update

        'clear screen
        DXFILL w1,RGB(0,0,0)
        'draw sprite
        DXDRAWALLSPRITES w1
        'show the changes
        DXFLIP w1
```

```
RETURN
```

Loads 3 sprites from the file: C:\Program Files\Ionic Wind\Creative BASIC\samples\mouth.bmp (change as needed), renders all the sprites to the screen with DXDRAWALLSPRITES

See Also

DXSPRITE;DXDRAWSPRITE

DXDRAWMAP

Usage

```
DXDRAWMAP winHandle {,transparency}
```

Parameters

winHandle	a window containing a map.
transparency	optional, transparency colour set using RGB().

Returns

nothing

Description

Draws the map to the screen back buffer and is then displayed using DXFLIP. Optional transparency color can be specified by specifiying an RGB value.

Examples

```
DEF myWindow:WINDOW
DEF state1,state2,spr1,spr2,x,y1,y2,bitmap,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@NOCAPTION,0,"",mainwindow
CREATESCREEN(myWindow,640,480,16)
SETFONT myWindow,"Courier New",12,700
DRAWMODE myWindow,@TRANSPARENT
FRONTPEN myWindow,RGB(255,255,0)

'load the tile images for the map
DXNEWMAP(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\tiles.bmp",64,64,0)
'create a blank map filled with tile#1 and scroll wrapped
DXCREATEMAP myWindow,10,10,1,1

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
CASE @IDCHAR
    SELECT @code
        CASE ASC("Q")
        CASE ASC("q")
            run = 0
```

```
    ENDSELECT
ENDSELECT
RETURN

SUB update
    'check for arrow keys and scroll appropriately
    if GETKEYSTATE(37) THEN DXSCROLLMAP(myWindow,@SCROLLLEFT,2)
    if GETKEYSTATE(38) THEN DXSCROLLMAP(myWindow,@SCROLLUP,2)
    if GETKEYSTATE(39) THEN DXSCROLLMAP(myWindow,@SCROLLRIGHT,2)
    if GETKEYSTATE(40) THEN DXSCROLLMAP(myWindow,@SCROLLDOWN,2)
    'draw the map to the back buffer
    DXDRAWMAP myWindow
    'draw a message
    MOVE myWindow,10,10
    PRINT myWindow,"Map Example - Press Q to quit "
    MOVE myWindow,10,25
    PRINT myWindow,"Use arrow keys to scroll around"
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads in tiles with DXNEWMAP then creates a blank map with DXCREATEMAP, and fills it with the tile. Tiles are loaded from C:\Tiles.bmp you can find the Tiles.bmp in your Creative BASIC samples folder, change this path to point to it. In the update subroutine the map is draw with DXDRAWMAP and a message is written over the top.

See Also

DXLOADMAP;DXCREATEMAP

DXDRAWSPRITE

Usage

```
DXDRAWSPRITE winhandle, sprite
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
sprite	the sprite to be drawn. must be loaded already.

Returns

Nothing.

Description

Draws the sprite referenced in the window to the back buffer of the screen.

Examples

```
DEF myWindow:WINDOW
DEF spr1,spr2,spr3:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@SIZE,0,"DXSprite",main
CREATESCREEN(myWindow,640,480)

'load the sprite
spr1 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,1,RGB(87,87,87))
spr2 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,2,RGB(87,87,87))
```

```
spr3 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,3,RGB(87,87,87))

DXMOVESPRITE spr1,100,100
DXMOVESPRITE spr2,200,200
DXMOVESPRITE spr3,300,300

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

main:
    SELECT @CLASS
        CASE @IDDXUPDATE
            GOSUB update
        CASE @IDCLOSEWINDOW
            run = 0
    ENDSELECT
RETURN

SUB update
    'clear screen
    DXFILL myWindow,RGB(0,0,0)
    'draw sprite
    DXDRAWSPRITE myWindow,spr2
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads in 3 sprites from `C:\Program Files\Ionic Wind\Creative BASIC\samples\mouth.bmp` (change as required), and uses `DXDRAWSPRITE` to draw `spr2` to the screen.

See Also

`DXSPRITE`;`DXDRAWALLSPRITES`

DXFILL

Usage

```
 DXFILL window, color
```

Parameters

`winhandle`	a user defined DirectX `WINDOW` variable.
`color`	the RGB color to fill the window with.

Returns

Nothing.

Description

Fills the back buffer of the screen attached to the window with the specified color. If the screen is full screen and 8 bpp then the color is a palette entry value otherwise the color is an RGB value.

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@SIZE,0,"DXFill",mainwindow
```

```
CREATESCREEN(myWindow,640,480)

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN

SUB update
    'clear screen
    DXFILL myWindow,RGB(40,40,40)

    FRONTPEN myWindow, RGB(0,0,255)
    MOVE myWindow,0,0
    PRINT myWindow,"DXFill"

    DXFLIP myWindow
RETURN
```

Sets up a window and colours it with `DXFILL` every `@IDDXUPDATE`.

See Also

DXFLIP

DXFLIP

Usage

```
DXFLIP winHandle {,fast} {,stretch}
```

Parameters

`winhandle`	a user defined `WINDOW` or `DIALOG` variable.
`fast`	optional. 0 = wait for the monitor vertical sync signal (vsync). 1 = do not wait for the vsync signal.
`stretch`	optional. 0 = do not enable stretching on scalable windows. 1 = enable scaling.

Returns

Nothing.

Description

Flips the back buffer of the DirectX screen to the foreground. Use when all drawing operations are completed to show the changes. `winHandle` must have a DirectX or Direct3D screen attached with the `CREATESCREEN` or `CREATE3DSCREEN` functions. If `fast` = 1 then the flip happens immediately, if = 0 or omitted then the flip waits for vertical sync of the monitor. When in windowed mode, `stretch` can be set to 0 to prevent the screen from being scaled to the size of the window.

Examples

```
DEF myWindow:WINDOW
```

```
DEF state:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW,0,"DXFlip",mainwindow

CREATESCREEN(myWindow,640,480)

DXFILL(myWindow,RGB(255,0,255))
DXFLIP myWindow
state = 0
STARTTIMER myWindow,500

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
    SELECT @CLASS
        CASE @IDCLOSEWINDOW
            run = 0
            STOPTIMER myWindow
        CASE @IDDXUPDATE
            IF(state=0) THEN DXFILL(myWindow,RGB(255,0,255)) ELSE
DXFILL(myWindow,RGB(0,255,0))
            REM FLIP CHANGES
            DXFLIP myWindow
        CASE @IDTIMER
            state = NOT state
    ENDSELECT
RETURN
```

Setups up a timer and flips the screen between two different colours.

See Also

DXFILL

DXGETMAPCOUNT

Usage

```
DXGETMAPCOUNT(window)
```

Parameters

`winhandle` a user defined `WINDOW` variable containing a map.

Returns

An integer.

Description

Returns the number of tiles stored in the bitmap loaded when the map was created.

Examples

```
DEF myWindow:WINDOW
DEF state1,state2,spr1,spr2,x,y1,y2,bitmap,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@NOCAPTION,0,"",main
'create the screen 16bpp fullscreen
```

```
CREATESCREEN(myWindow,640,480,16)
SETFONT myWindow,"Courier New",12,700
DRAWMODE myWindow,@TRANSPARENT
FRONTPEN myWindow,RGB(200,200,0)

'load the tile images for the map
DXNEWMAP(myWindow,GETSTARTPATH + "C:\mapdata.jpg",64,64,42)
'load the map definition file, no scrollwrap
DXLOADMAP(myWindow,GETSTARTPATH + "C:\level.map")
'Move to the start of the map
DXMOVEMAP myWindow,0,0

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

main:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
CASE @IDCHAR
    SELECT @code
        CASE ASC("Q")
        CASE ASC("q")
            run = 0
    ENDSELECT
ENDSELECT
RETURN

SUB update
    'check for arrow keys and scroll appropriately
    if GETKEYSTATE(37) THEN DXSCROLLMAP(myWindow,@SCROLLLEFT,2)
    if GETKEYSTATE(38) THEN DXSCROLLMAP(myWindow,@SCROLLUP,2)
    if GETKEYSTATE(39) THEN DXSCROLLMAP(myWindow,@SCROLLRIGHT,2)
    if GETKEYSTATE(40) THEN DXSCROLLMAP(myWindow,@SCROLLDOWN,2)
    'draw the map to the back buffer
    DXDRAWMAP myWindow
    MOVE myWindow,10,10
    PRINT myWindow,"Map Example - Press Q to quit "
    MOVE myWindow,10,25
    PRINT myWindow,"Use arrow keys to scroll around"
```

```
    MOVE myWindow,10,40
    PRINT "Map count: ",DXGETMAPCOUNT(myWindow)
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads in the tiles with DXNEWMAP then loads in the map definitions with DXLOADMAP. Tiles are loaded from C:\mapdata.jpg which can be found inside your Creative BASIC samples folder along with the C:\level.map file which is required for DXLOADMAP. Prints to the screen DXMAPCOUNT, which is the number of tiles stored in the bitmap.

See Also

DXGETMAPTILE;DXGETMAPHEIGHT;DXGETMAPWIDTH

DXGETMAPHEIGHT()

Usage

```
DXGETMAPHEIGHT(winhandle)
```

Parameters

winhandle — a user defined WINDOW variable containing a map.

Returns

An integer.

Description

Returns the height of the map in tiles (not pixels).

Examples

```
DEF myWindow:WINDOW
DEF state1,state2,spr1,spr2,x,y1,y2,bitmap,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@NOCAPTION,0,"",mainwindow
'create the screen 16bpp fullscreen
CREATESCREEN(myWindow,640,480,16)
SETFONT myWindow,"Courier New",12,700
DRAWMODE myWindow,@TRANSPARENT
FRONTPEN myWindow,RGB(200,200,0)

'load the tile images for the map
DXNEWMAP(myWindow,"C:\mapdata.jpg",64,64,42)
'load the map definition file, no scrollwrap
DXLOADMAP(myWindow,"C:\level.map")
'Move to the start of the map
DXMOVEMAP myWindow,0,0

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
    SELECT @CLASS
        CASE @IDDXUPDATE
            GOSUB update
```

```
        CASE @IDCLOSEWINDOW
            run = 0
    CASE @IDCHAR
        SELECT @code
            CASE ASC("Q")
            CASE ASC("q")
                run = 0
        ENDSELECT
    ENDSELECT
RETURN

SUB update
    'check for arrow keys and scroll appropriately
    if GETKEYSTATE(37) THEN DXSCROLLMAP(myWindow,@SCROLLLEFT,2)
    if GETKEYSTATE(38) THEN DXSCROLLMAP(myWindow,@SCROLLUP,2)
    if GETKEYSTATE(39) THEN DXSCROLLMAP(myWindow,@SCROLLRIGHT,2)
    if GETKEYSTATE(40) THEN DXSCROLLMAP(myWindow,@SCROLLDOWN,2)
    'draw the map to the back buffer
    DXDRAWMAP myWindow
    MOVE myWindow,10,10
    PRINT myWindow,"Map Example - Press Q to quit "
    MOVE myWindow,10,25
    PRINT myWindow,"Use arrow keys to scroll around"
    MOVE myWindow,10,40
    PRINT "Map Height: ",DXGETMAPHEIGHT(myWindow)
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads in the tiles with DXNEWMAP then loads in the map definitions with DXLOADMAP. The required files for the example are loaded from the default samples directory for Creative BASIC. The example prints to the screen DXMAPHEIGHT, which is the height of the map in tiles.

See Also

DXGETMAPTILE;DXGETMAPCOUNT;DXGETMAPWIDTH

DXGETMAPTILE()

Usage

```
DXGETMAPTILE( window,x,y )
```

Parameters

winhandle	a user defined WINDOW variable containing a map.
x	map x coordinate
y	map y coordinate

Returns

An integer.

Description

Returns the tile number located at map coordinates x and y.

Examples

```
DEF myWindow:WINDOW
DEF state1,state2,spr1,spr2,x,y1,y2,bitmap,delay:INT
```

```
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@NOCAPTION,0,"",main
'create the screen 16bpp fullscreen
CREATESCREEN(myWindow,640,480,16)
SETFONT myWindow,"Courier New",12,700
DRAWMODE myWindow,@TRANSPARENT
FRONTPEN myWindow,RGB(200,200,0)

'load the tile images for the map
DXNEWMAP(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mapdata.jpg",64,64,42)
'load the map definition file, no scrollwrap
DXLOADMAP(myWindow,"C:\Program Files\Ionic Wind\Creative BASIC\samples\level.map")
'Move to the start of the map
DXMOVEMAP myWindow,0,0

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

main:
    SELECT @CLASS
        CASE @IDDXUPDATE
            GOSUB update
        CASE @IDCLOSEWINDOW
            run = 0
        CASE @IDCHAR
            SELECT @code
                CASE ASC("Q")
                CASE ASC("q")
                    run = 0
            ENDSELECT
    ENDSELECT
RETURN

SUB update
    'check for arrow keys and scroll appropriately
    if GETKEYSTATE(37) THEN DXSCROLLMAP(myWindow,@SCROLLLEFT,2)
    if GETKEYSTATE(38) THEN DXSCROLLMAP(myWindow,@SCROLLUP,2)
    if GETKEYSTATE(39) THEN DXSCROLLMAP(myWindow,@SCROLLRIGHT,2)
    if GETKEYSTATE(40) THEN DXSCROLLMAP(myWindow,@SCROLLDOWN,2)
    'clear the screen (avoids corrupton with missing/transparent tiles).
    DXFILL myWindow,RGB(0,0,20)
    'draw the map to the back buffer
    DXDRAWMAP myWindow
```

```
    MOVE myWindow,10,10
    PRINT myWindow,"Map Example - Press Q to quit "
    MOVE myWindow,10,25
    PRINT myWindow,"Use arrow keys to scroll around"
    MOVE myWindow,10,40
    PRINT "Map Tile: ",DXGETMAPTILE(myWindow,0,0)
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads in the tiles with DXNEWMAP then loads in the map definitions with DXLOADMAP. The required files for the example are loaded from the default samples directory for Creative BASIC. In this example, the program prints to the screen the map tile index of the tile at map position 0, 0 (using DXGETMAPTILE).

See Also

DXGETMAPCOUNT;DXGETMAPWIDTH;DXGETMAPHEIGHT

DXGETMAPWIDTH()

Usage

```
DXGETMAPWIDTH window
```

Parameters

winhandle — a user defined WINDOW variable containing a map.

Returns

An integer.

Description

Returns the width of the map in tiles.

Examples

```
DEF myWindow:WINDOW
DEF state1,state2,spr1,spr2,x,y1,y2,bitmap,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@NOCAPTION,0,"",mainwindow
'create the screen 16bpp fullscreen
CREATESCREEN(myWindow,640,480,16)
SETFONT myWindow,"Courier New",12,700
DRAWMODE myWindow,@TRANSPARENT
FRONTPEN myWindow,RGB(200,200,0)

'load the tile images for the map
DXNEWMAP(myWindow,GETSTARTPATH + "C:\mapdata.jpg",64,64,42)
'load the map definition file, no scrollwrap
DXLOADMAP(myWindow,GETSTARTPATH + "C:\level.map")
'Move to the start of the map
DXMOVEMAP myWindow,0,0

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END
```

```
mainwindow:
    SELECT @CLASS
        CASE @IDDXUPDATE
            GOSUB update
        CASE @IDCLOSEWINDOW
            run = 0
    CASE @IDCHAR
        SELECT @code
            CASE ASC("Q")
            CASE ASC("q")
                run = 0
        ENDSELECT
    ENDSELECT
RETURN

SUB update
    'check for arrow keys and scroll appropriately
    if GETKEYSTATE(37) THEN DXSCROLLMAP(myWindow,@SCROLLLEFT,2)
    if GETKEYSTATE(38) THEN DXSCROLLMAP(myWindow,@SCROLLUP,2)
    if GETKEYSTATE(39) THEN DXSCROLLMAP(myWindow,@SCROLLRIGHT,2)
    if GETKEYSTATE(40) THEN DXSCROLLMAP(myWindow,@SCROLLDOWN,2)
    'draw the map to the back buffer
    DXDRAWMAP myWindow
    MOVE myWindow,10,10
    PRINT myWindow,"Map Example - Press Q to quit "
    MOVE myWindow,10,25
    PRINT myWindow,"Use arrow keys to scroll around"
    MOVE myWindow,10,40
    PRINT "Map Width: ",DXGETMAPWIDTH(myWindow)
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads in the tiles with DXNEWMAP then loads in the map definitions with DXLOADMAP. The required files for the example are loaded from the default samples directory for Creative BASIC. The example prints to the screen DXMAPWIDTH, which is the width of the map in tiles.

See Also

DXGETMAPTILE;DXGETMAPCOUNT;DXGETMAPHEIGHT

DXGETSPRITEDATA

Usage

```
DXGETSPRITEDATA(sprite, data-ID)
```

Parameters

spriteID	a handle to a user defined sprite variable.
data-ID	See appendix B (System Variables and Constants) for a list of data-ID's.

Returns

sprite value

Description

Returns the value specified from the sprite. Sprite must have been successfully created with the DXSPRITE function. See Appendix B (system variables and constants) for a list of data-ID's

Examples

```
DEF myWindow:WINDOW
DEF spr,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@SIZE,0,"DXSprite",mainwindow

CREATESCREEN(myWindow,640,480)

'load the sprite
spr =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,3,RGB(87,87,87))

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN

SUB update
    'animate the sprite
    delay = delay + 1
    IF delay > 10
        DXSETSPRITEDATA spr,@SDFRAME,DXGETSPRITEDATA(spr,@SDFRAME) + 1
        IF DXGETSPRITEDATA(spr,@SDFRAME) > 2 THEN DXSETSPRITEDATA spr,@SDFRAME,0
        delay = 0
    ENDIF

    'clear screen
    DXFILL myWindow,RGB(0,0,0)
    'draw sprite
    DXDRAWALLSPRITES myWindow
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads in a sprite from C:\Program Files\Ionic Wind\Creative BASIC\samples\mouth.bmp (change path

as required), and animates it using DXSETSPRITEDATA and DXGETSPRITEDATA along with the @SDFRAME system constant.

See Also

DXSETSPRITEDATA;DXSPRITE

DXHITANY( )

Usage

```
DXHITANY(window, sprite1 {,pixels})
```

Parameters

winhandle	a user defined WINDOW variable containing sprites.
sprite1	the sprite to check collisions for.
pixels	optional. How many pixels to check collision by.

Returns

Integer reference of collided sprite

Description

Tests all of the sprites in the window to see if any of them intersect with sprite1. Returns the integer identifier of the first hit sprite.

Examples

```
DEF myWindow:WINDOW
DEF spr1,spr2,spr3,spriteHit:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@SIZE,0,"DXSprite",mainwindow

CREATESCREEN(myWindow,640,480)

'load the sprite
spr1 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,1,RGB(87,87,87))
spr2 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,2,RGB(87,87,87))
spr3 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,3,RGB(87,87,87))

DXMOVESPRITE spr1,200,200
DXMOVESPRITE spr2,300,300
DXMOVESPRITE spr3,210,210

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
```

```
        run = 0
ENDSELECT
RETURN

SUB update
    'clear screen
    DXFILL myWindow,RGB(0,0,0)
    spriteHit = DXHITANY(myWindow,spr1)

        FRONTPEN myWindow, RGB(0,0,255)
        MOVE myWindow,0,0
        PRINT myWindow,"COLLISION! with ", spriteHit

    'draw sprite
    DXDRAWALLSPRITES myWindow
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads 3 sprites from `C:\Program Files\Ionic Wind\Creative BASIC\samples\mouth.bmp` (change as required), checks sprite 1 for collisions with `DXHITANY` and prints which sprite it has collided with.

See Also

DXSPRITE;DXHITSPRITE

DXHITSPRITE( )

Usage

```
DXHITSPRITE( sprite1,sprite2 {,pixels} )
```

Parameters

sprite1	a sprite to check collisions for.
sprite2	a sprite to check collisions against.
pixels	optional How many pixels to check for collision by.

Returns

True on collision.

Description

Tests whether any part of sprite2 intersects sprite1. Returns 1 (TRUE) if the sprites are touching.

Examples

```
DEF myWindow:WINDOW
DEF spr1,spr2:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@SIZE,0,"DXSprite",mainwindow

CREATESCREEN(myWindow,640,480)

'load the sprite
spr1 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,1,RGB(87,87,87))
spr2 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,2,RGB(87,87,87))
```

```
DXMOVESPRITE spr1,200,200
DXMOVESPRITE spr2,210,210

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
    SELECT @CLASS
        CASE @IDDXUPDATE
            GOSUB update
        CASE @IDCLOSEWINDOW
            run = 0
    ENDSELECT
RETURN

SUB update
    'clear screen
    DXFILL myWindow,RGB(0,0,0)

    IF DXHITSPRITE(spr1,spr2) = 1
        FRONTPEN myWindow, RGB(0,0,255)
        MOVE myWindow,0,0
        PRINT myWindow,"COLLISION!"
    ENDIF

    'draw sprite
    DXDRAWALLSPRITES myWindow
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads 2 sprites from `C:\Program Files\Ionic Wind\Creative BASIC\samples\mouth.bmp` (change as required), and places them on the screen so they are overlapping each other. The program checks for sprite collision with `DXHITSPRITE` and prints a "COLLISION!" message to the screen.

See Also

`DXSPRITE`;`DXHITANY`;`DXHITSPRITETILE`

DXHITSPRITETILE( )

Usage

```
DXHITSPRITETILE( winhandle, sprite, tile )
```

Parameters

`winhandle`	a user defined `WINDOW` variable, set as a DirectX screen.
`sprite`	sprite to check collisions for.
`tile`	tile to check collisions with.

Returns

0 or 1.

Description

Test whether any part of the sprite intersects a tile in a map. Returns 1 if the sprite is touching the tile.

Examples

```
DEF myWindow:WINDOW
DEF spr,spritehit,x,y:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@NOCAPTION,0,"",mainwindow
CREATESCREEN(myWindow,640,480,16)
SETFONT myWindow,"Courier New",12,700

x=360
y=360

DRAWMODE myWindow,@TRANSPARENT
FRONTPEN myWindow,RGB(255,255,0)

'load the tile images for the map
DXNEWMAP(myWindow,"C:\tiles.bmp",64,64,0)
'create a blank map filled with tile#1 and scroll wrapped
DXCREATEMAP myWindow,10,10,1,1
'set top left map tile to 0
DXSETMAPTILE myWindow,0,0,0

spr =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,3,RGB(87,87,87))

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
CASE @IDCHAR
    SELECT @code
        CASE ASC("Q")
        CASE ASC("q")
            run = 0
    ENDSELECT
ENDSELECT
RETURN
```

```
SUB update
    DXFILL myWindow,RGB(0,0,0)
    'check for arrow keys and move appropriately
    IF GETKEYSTATE(37) THEN x = x - 1
    IF GETKEYSTATE(38) THEN y = y - 1
    IF GETKEYSTATE(39) THEN x = x + 1
    IF GETKEYSTATE(40) THEN y = y + 1

    DXMOVESPRITE(spr,x,y)

    'draw the map to the back buffer
    DXDRAWMAP myWindow
    DXDRAWALLSPRITES myWindow
    spriteHit = DXHITSPRITETILE(myWindow,spr,0)
    IF spriteHit > 0
        FRONTPEN myWindow, RGB(0,0,255)
        MOVE myWindow,0,0
        PRINT myWindow,"COLLISION! with ", spriteHit
        spriteHit = 0
    ENDIF
    'draw a message
    MOVE myWindow,10,10
    PRINT myWindow,"Map Example - Press Q to quit "
    MOVE myWindow,10,25
    PRINT myWindow,"Use arrow keys to move around"
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads tile images with DXNEWMAP, tiles are loaded from C:\Tiles.bmp you can find the Tiles.bmp in your Creative BASIC samples folder, change this path to point to it. Sets the map tile at 0,0 to 0 (a black square), then loads a sprite with DXSPRITE and checks for collisions with the black tile with DXHITSPRITETILE.

See Also

DXHITANY; DXHITSPRITE

DXLOADMAP

Usage

```
DXLOADMAP winhandle, mapfile {,scrollwrap}
```

Parameters

winhandle	a user defined WINDOW variable, set as a DirectX screen.
mapfile	map to load in.
scrollwrap	optional, whether the tile should be wrapped.

Returns

Nothing.

Description

Loads a map definition file and creates a map in the screen. Map must have been created with the DXNEWMAP function before this function is called. Optional: scrollwrap = 1 to wrap the images when the map is moved past the boundaries of the screen. Returns 0 on failure.

Examples

```
DEF myWindow:WINDOW
DEF state1,state2,spr1,spr2,x,y1,y2,bitmap,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@NOCAPTION,0,"",mainwindow
'create the screen 16bpp fullscreen
CREATESCREEN(myWindow,640,480,16)
SETFONT myWindow,"Courier New",12,700
DRAWMODE myWindow,@TRANSPARENT
FRONTPEN myWindow,RGB(200,200,0)

'load the tile images for the map
DXNEWMAP(myWindow,"C:\mapdata.jpg",64,64,42)
'load the map definition file, no scrollwrap
DXLOADMAP(myWindow,"C:\level.map")
'Move to the start of the map
DXMOVEMAP myWindow,0,0

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
CASE @IDCHAR
    SELECT @code
        CASE ASC("Q")
        CASE ASC("q")
            run = 0
    ENDSELECT
ENDSELECT
RETURN

SUB update
    'check for arrow keys and scroll appropriately
    IF GETKEYSTATE(37) THEN DXSCROLLMAP(myWindow,@SCROLLLEFT,2)
    IF GETKEYSTATE(38) THEN DXSCROLLMAP(myWindow,@SCROLLUP,2)
    IF GETKEYSTATE(39) THEN DXSCROLLMAP(myWindow,@SCROLLRIGHT,2)
```

```
    IF GETKEYSTATE(40) THEN DXSCROLLMAP(myWindow,@SCROLLDOWN,2)
    'draw the map to the back buffer
    DXDRAWMAP myWindow
    MOVE myWindow,10,10
    PRINT myWindow,"Map Example - Press Q to quit "
    MOVE myWindow,10,25
    PRINT myWindow,"Use arrow keys to scroll around"
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads in the tiles with DXNEWMAP then loads in the map definitions with DXLOADMAP. Tiles are loaded from C:\mapdata.jpg, this file can be found inside your Creative BASIC samples folder along with the C:\level.map file which is required for DXLOADMAP.

See Also

DXNEWMAP; DXCREATEMAP

DXLOADPALETTE

Usage

```
 DXLOADPALETTE winhandle, bitmapfile
```

Parameters

winhandle	a window set as a direxT X screen.
bitmapfile	bitmap to create palette from.

Returns

Nothing.

Description

Creates a palette for 8 bit screens from the bitmap specified. A palette must be loaded for transparency to function correctly in 8 bpp. The function should not be used for higher screen bpp's.

Examples

```
DEF myWindow:WINDOW
DEF spr,palettecolor:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@SIZE,0,"DXSprite",mainwindow

CREATESCREEN(myWindow,640,480,8)

'load the sprite
DXLOADPALETTE(myWindow, "C:\bug.bmp")
for palettecolor = 2 to 255
DXSETCOLOR myWindow, palettecolor,RGB(32,32,32)
next palettecolor
spr =  DXSPRITE(myWindow,"C:\bug.bmp",84,80,1)

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
```

```
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
    CASE @IDCHAR
        SELECT @code
            CASE ASC("Q")
            CASE ASC("q")
                run = 0
        ENDSELECT
ENDSELECT
RETURN

SUB update
    'clear screen
    DXFILL myWindow,RGB(0,0,0)
    'draw sprite
    DXDRAWALLSPRITES myWindow
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads a palette for a window named myWindow from the file C:\bug.bmp and changes the palette colors to grey except for a few colors. This is why the image will look strange.

DXMOVEMAP

Usage

```
DXMOVEMAP winhandle ,x ,y
```

Parameters

winhandle	a user defined WINDOW variable.
x	x pixel position to move to.
y	y pixel position to move to.

Returns

Nothing.

Description

Moves the map to the pixel position specified. The position will be displayed starting at the upper left corner of the screen.

Examples

```
DEF myWindow:WINDOW
DEF state1,state2,spr1,spr2,x,y1,y2,bitmap,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@NOCAPTION,0,"",mainwindow
'create the screen 16bpp fullscreen
CREATESCREEN(myWindow,640,480,16)
SETFONT myWindow,"Courier New",12,700
DRAWMODE myWindow,@TRANSPARENT
FRONTPEN myWindow,RGB(200,200,0)
```

```
'load the tile images for the map
DXNEWMAP(myWindow,"C:\mapdata.jpg",64,64,42)
'load the map definition file, no scrollwrap
DXLOADMAP(myWindow,GETSTARTPATH + "C:\level.map")
'Move to the start of the map
DXMOVEMAP myWindow,0,0

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
CASE @IDCHAR
    SELECT @code
        CASE ASC("Q")
        CASE ASC("q")
            run = 0
    ENDSELECT
ENDSELECT
RETURN

SUB update
    'check for arrow keys and scroll appropriately
    if GETKEYSTATE(37) THEN DXSCROLLMAP(myWindow,@SCROLLLEFT,2)
    if GETKEYSTATE(38) THEN DXSCROLLMAP(myWindow,@SCROLLUP,2)
    if GETKEYSTATE(39) THEN DXSCROLLMAP(myWindow,@SCROLLRIGHT,2)
    if GETKEYSTATE(40) THEN DXSCROLLMAP(myWindow,@SCROLLDOWN,2)
    'draw the map to the back buffer
    DXDRAWMAP myWindow
    MOVE myWindow,10,10
    PRINT myWindow,"Map Example - Press Q to quit "
    MOVE myWindow,10,25
    PRINT myWindow,"Use arrow keys to scroll around"
    'show the changes
    DXFLIP myWindow
RETURN
```

A map is created and the tile set loaded, then DXMOVEMAP is used to move to the start of the map at position 0,0.

See Also

DXNEWMAP; DXLOADMAP; DXCREATEMAP

DXMOVESPRITE

Usage

```
DXMOVESPRITE sprite, x, y
```

Parameters

sprite	a sprite.
x	the x coordinate to move the sprite to.
y	the y coordinate to move the sprite to.

Returns

Nothing.

Description

Moves the sprite to the position specified by x and y. Coordinates are specified in screen pixels.

Examples

```
DEF myWindow:WINDOW
DEF spr1,spr2,spr3:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@SIZE,0,"DXSprite",mainwindow


CREATESCREEN(myWindow,640,480)


'load the sprite
spr1 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,1,RGB(87,87,87))
spr2 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,2,RGB(87,87,87))
spr3 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,3,RGB(87,87,87))


DXMOVESPRITE spr1,100,100
DXMOVESPRITE spr2,200,200
DXMOVESPRITE spr3,300,300


run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END


mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

```
SUB update
    'clear screen
    DXFILL myWindow,RGB(0,0,0)
    'draw sprite
    DXDRAWALLSPRITES myWindow
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads in 3 sprites from C:\Program Files\Ionic Wind\Creative BASIC\samples\mouth.bmp (change as required), and displays them in a diagonal line using DXMOVESPRITE.

See Also

DXSPRITE

DXNEWMAP

Usage

```
DXNEWMAP winhandle, bitmapfile, tilewidth,tileheight,tiles
```

Parameters

winhandle	a user defined WINDOW variable.
bitmapfile	the file to load from.
tilewidth	tile width.
tileheight	tile height.
tiles	number of tiles.

Returns

0 on failure.

Description

Loads a new set of tiles for the map in the window. width and height specifiy the width and height of each tile and frames specifies the number of tiles in the image. If frames = 0 then the number of tiles is calculated by Creative BASIC. Returns 0 on failure.

Examples

```
DEF myWindow:WINDOW
DEF state1,state2,spr1,spr2,x,y1,y2,bitmap,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@NOCAPTION,0,"",mainwindow
CREATESCREEN(myWindow,640,480,16)
SETFONT myWindow,"Courier New",12,700
DRAWMODE myWindow,@TRANSPARENT
FRONTPEN myWindow,RGB(255,255,0)

'load the tile images for the map
DXNEWMAP(myWindow,"C:\tiles.bmp",64,64,0)
'create a blank map filled with tile#1 and scroll wrapped
DXCREATEMAP myWindow,10,10,1,1

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END
```

```
mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
CASE @IDCHAR
    SELECT @code
        CASE ASC("Q")
        CASE ASC("q")
            run = 0
    ENDSELECT
ENDSELECT
RETURN

SUB update
    'check for arrow keys and scroll appropriately
    IF GETKEYSTATE(37) THEN DXSCROLLMAP(myWindow,@SCROLLLEFT,2)
    IF GETKEYSTATE(38) THEN DXSCROLLMAP(myWindow,@SCROLLUP,2)
    IF GETKEYSTATE(39) THEN DXSCROLLMAP(myWindow,@SCROLLRIGHT,2)
    IF GETKEYSTATE(40) THEN DXSCROLLMAP(myWindow,@SCROLLDOWN,2)
    'draw the map to the back buffer
    DXDRAWMAP myWindow
    'draw a message
    MOVE myWindow,10,10
    PRINT myWindow,"Map Example - Press Q to quit "
    MOVE myWindow,10,25
    PRINT myWindow,"Use arrow keys to scroll around"
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads tile images with DXNEWMAP, tiles are loaded from C:\Tiles.bmp you can find the Tiles.bmp in your Creative BASIC samples folder, change this path to point to it.

See Also

DXCREATEMAP; DXLOADMAP

DXREMOVESPRITE

Usage

```
DXREMOVESPRITE winhandle, sprite
```

Parameters

winhandle	a user defined WINDOW variable.
sprite	a pre-loaded sprite

Returns

Nothing.

Description

Removes the sprite from the window and sets the sprite variable to 0. Sprite must be reloaded with DXSPRITE to be used again.

Examples

```
DEF myWindow:WINDOW
DEF spr1,spr2,spr3,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@SIZE,0,"DXSprite",mainwindow

CREATESCREEN(myWindow,640,480)

'load the sprite
spr1 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,1,RGB(87,87,87))
spr2 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,2,RGB(87,87,87))
spr3 =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,3,RGB(87,87,87))

DXMOVESPRITE spr1,100,100
DXMOVESPRITE spr2,200,200
DXMOVESPRITE spr3,300,300

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN

SUB update

    delay = delay + 1
    IF delay = 100
        DXREMOVESPRITE myWindow,spr2
    ENDIF

    'clear screen
    DXFILL myWindow,RGB(0,0,0)
    'draw sprite
    DXDRAWALLSPRITES myWindow
    'show the changes
```

```
    DXFLIP myWindow
RETURN
```

Loads in 3 sprites using C:\Program Files\Ionic Wind\Creative BASIC\samples\mouth.bmp (change as required), after a short delay one of the sprites is removed with DXREMOVESPRITE.

See Also

DXSPRITE

DXSAVEMAP

Usage

```
 DXSAVEMAP winhandle, filename
```

Parameters

winhandle	a user defined WINDOW variable.
filename	filename to save to

Returns

Nothing.

Description

Saves a map definition file to the file specified.

Examples

```
DEF myWindow:WINDOW
DEF state1,state2,spr1,spr2,x,y1,y2,bitmap,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@NOCAPTION,0,"",mainwindow
CREATESCREEN(myWindow,640,480,16)
SETFONT myWindow,"Courier New",12,700
DRAWMODE myWindow,@TRANSPARENT
FRONTPEN myWindow,RGB(255,255,0)

'load the tile images for the map
DXNEWMAP(myWindow,"C:\tiles.bmp",64,64,0)
'create a blank map filled with tile#1 and scroll wrapped
DXCREATEMAP myWindow,10,10,1,1
DXSAVEMAP(myWindow,"C:\savedmap.map")

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
CASE @IDCHAR
    SELECT @code
        CASE ASC("Q")
        CASE ASC("q")
```

```
            run = 0
    ENDSELECT
ENDSELECT
RETURN

SUB update
    'check for arrow keys and scroll appropriately
    IF GETKEYSTATE(37) THEN DXSCROLLMAP(myWindow,@SCROLLLEFT,2)
    IF GETKEYSTATE(38) THEN DXSCROLLMAP(myWindow,@SCROLLUP,2)
    IF GETKEYSTATE(39) THEN DXSCROLLMAP(myWindow,@SCROLLRIGHT,2)
    IF GETKEYSTATE(40) THEN DXSCROLLMAP(myWindow,@SCROLLDOWN,2)
    'draw the map to the back buffer
    DXDRAWMAP myWindow
    'draw a message
    MOVE myWindow,10,10
    PRINT myWindow,"Map Example - Press Q to quit "
    MOVE myWindow,10,25
    PRINT myWindow,"Use arrow keys to scroll around"
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads in the tiles with DXNEWMAP then loads in the map definitions with DXLOADMAP. Tiles are loaded from C:\mapdata.jpg, this file can be found inside your Creative BASIC samples folder along with the C:\level.map file which is required for DXLOADMAP. The map is then saved to C:\savedmap.map with DXSAVEMAP.

See Also

DXLOADMAP

DXSCROLLMAP

Usage

```
DXSCROLLMAP winhandle, direction, amount
```

Parameters

winhandle	a user defined WINDOW variable.
direction	the direction to scroll (@SCROLLLEFT, @SCROLLUP etc).
amount	the amount to scroll by.

Returns

Nothing.

Description

Scrolls the map in the window. Direction is one of the predefined flags:

@SCROLLUP, @SCROLLDOWN, @SCROLLLEFT or @SCROLLRIGHT

Examples

```
DEF myWindow:WINDOW
DEF state1,state2,spr1,spr2,x,y1,y2,bitmap,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@NOCAPTION,0,"",mainwindow
CREATESCREEN(myWindow,640,480,16)
SETFONT myWindow,"Courier New",12,700
```

```
DRAWMODE myWindow,@TRANSPARENT
FRONTPEN myWindow,RGB(255,255,0)

'load the tile images for the map
DXNEWMAP(myWindow,"C:\tiles.bmp",64,64,0)
'create a blank map filled with tile#1 and scroll wrapped
DXCREATEMAP myWindow,10,10,1,1

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
CASE @IDCHAR
    SELECT @code
        CASE ASC("Q")
        CASE ASC("q")
            run = 0
    ENDSELECT
ENDSELECT
RETURN

SUB update
    'check for arrow keys and scroll appropriately
    IF GETKEYSTATE(37) THEN DXSCROLLMAP(myWindow,@SCROLLLEFT,2)
    IF GETKEYSTATE(38) THEN DXSCROLLMAP(myWindow,@SCROLLUP,2)
    IF GETKEYSTATE(39) THEN DXSCROLLMAP(myWindow,@SCROLLRIGHT,2)
    IF GETKEYSTATE(40) THEN DXSCROLLMAP(myWindow,@SCROLLDOWN,2)
    'draw the map to the back buffer
    DXDRAWMAP myWindow
    'draw a message
    MOVE myWindow,10,10
    PRINT myWindow,"Map Example - Press Q to quit "
    MOVE myWindow,10,25
    PRINT myWindow,"Use arrow keys to scroll around"
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads tile images with DXNEWMAP, tiles are loaded from C:\Tiles.bmp you can find the Tiles.bmp in your Creative BASIC samples folder, change this path to point to it. Map is scrolled with DXSCROLLMAP when the arrow keys are pressed.

See Also

DXMOVEMAP

DXSETCOLOR

Usage

```
DXSETCOLOR winhandle, palette#, color
```

Parameters

winhandle	a user defined WINDOW variable.
pallete#	the palette number of the color you want to change.
color	the value of the color you want to replace color with.

Returns

Nothing.

Description

Changes the color of a palette entry. Should only be used with screens of 8 bpp or less.

Examples

```
DEF myWindow:WINDOW
DEF spr,palettecolor:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@SIZE,0,"DXSprite",mainwindow

CREATESCREEN(myWindow,640,480,8)

'load the sprite
DXLOADPALETTE(myWindow, "C:\Program Files\Ionic Wind\Creative BASIC\samples\bug.bmp")
for palettecolor = 2 to 255
DXSETCOLOR myWindow, palettecolor,RGB(32,32,32)
next palettecolor
spr =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\bug.bmp",84,80,1)

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
    CASE @IDCHAR
        SELECT @code
            CASE ASC("Q")
            CASE ASC("q")
                run = 0
```

```
        ENDSELECT
ENDSELECT
RETURN

SUB update
    'clear screen
    DXFILL myWindow,RGB(0,0,0)
    'draw sprite
    DXDRAWALLSPRITES myWindow
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads a palette for a window named myWindow from the file C:\bug.bmp and changes the palette colors to grey except for a few colors. This is why the image will look strange.

See Also

DXLOADPALLETE

DXSETMAPTILE

Usage

```
DXSETMAPTILE winhandle,x,y,tile
```

Parameters

winhandle	a user defined WINDOW variable.
x	map x coordinate.
y	map y coordinate.
tile	tile to change to.

Returns

Nothing.

Description

Changes the tile at location x, y to the tile specified. Tile is a positive integer that represents the tile in the bitmap file. X and Y are tile coordinates in the map

Examples

```
DEF myWindow:WINDOW
DEF state1,state2,spr1,spr2,x,y1,y2,bitmap,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@NOCAPTION,0,"",mainwindow
CREATESCREEN(myWindow,640,480,16)
SETFONT myWindow,"Courier New",12,700
DRAWMODE myWindow,@TRANSPARENT
FRONTPEN myWindow,RGB(255,255,0)

'load the tile images for the map
DXNEWMAP(myWindow,"C:\tiles.bmp",64,64,0)
'create a blank map filled with tile#1 and scroll wrapped
DXCREATEMAP myWindow,10,10,1,1
'set top left map tile to 0
DXSETMAPTILE myWindow,0,0,0

run = 1
WAITUNTIL run = 0
```

```
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
CASE @IDCHAR
    SELECT @code
        CASE ASC("Q")
        CASE ASC("q")
            run = 0
    ENDSELECT
ENDSELECT
RETURN

SUB update
    'check for arrow keys and scroll appropriately
    IF GETKEYSTATE(37) THEN DXSCROLLMAP(myWindow,@SCROLLLEFT,2)
    IF GETKEYSTATE(38) THEN DXSCROLLMAP(myWindow,@SCROLLUP,2)
    IF GETKEYSTATE(39) THEN DXSCROLLMAP(myWindow,@SCROLLRIGHT,2)
    IF GETKEYSTATE(40) THEN DXSCROLLMAP(myWindow,@SCROLLDOWN,2)
    'draw the map to the back buffer
    DXDRAWMAP myWindow
    'draw a message
    MOVE myWindow,10,10
    PRINT myWindow,"Map Example - Press Q to quit "
    MOVE myWindow,10,25
    PRINT myWindow,"Use arrow keys to scroll around"
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads tile images with DXNEWMAP, tiles are loaded from C:\Tiles.bmp you can find the Tiles.bmp in your Creative BASIC samples folder, change this path to point to it. Sets the map tile at 0,0 to 0 (a black square).

See Also

DXLOADMAP; DXCREATEMAP

DXSETSPRITEDATA

Usage

```
DXSETSPRITEDATA sprite, data-ID, value
```

Parameters

sprite	a sprite.
data-ID	see the section on DXGETSPRITEDATA/DXSETSPRITEDATA flags Appendix B

(system variables and constants) for a list of data-IDs.

`value` value for the sprite.

Returns

Nothing.

Description

Sets the value specified for the sprite. Sprite must have been successfully created with the `DXSPRITE` function. See Appendix B (system variables and constants) for a list of data-ID's.

Examples

```
DEF myWindow:WINDOW
DEF spr,delay:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@SIZE,0,"DXSetSpriteData",mainwindow

CREATESCREEN(myWindow,640,480)

'load the sprite
spr =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,3,RGB(87,87,87))

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN

SUB update
    'animate the sprite
    delay = delay + 1
    IF delay > 10
        DXSETSPRITEDATA spr,@SDFRAME,DXGETSPRITEDATA(spr,@SDFRAME) + 1
        IF DXGETSPRITEDATA(spr,@SDFRAME) > 2 THEN DXSETSPRITEDATA spr,@SDFRAME,0
        delay = 0
    ENDIF

    'clear screen
    DXFILL myWindow,RGB(0,0,0)
    'draw sprite
    DXDRAWALLSPRITES myWindow
    'show the changes
```

```
    DXFLIP myWindow
RETURN
```

Loads in a sprite (from C:\Program Files\Ionic Wind\Creative BASIC\samples\mouth.bmp - change as needed) and animates it using @SDFRAME.

See Also

DXSPRITE; DXGETSPRITEDATA

DXSPRITE

Usage

```
DXSPRITE winhandle, bitmapfile, width, height, frames {,transparency}
```

Parameters

winhandle	user defined WINDOW variable.
bitmapfile	bmp file to load sprite from.
width	width of one image tile.
height	height of one image tile.
frames	number of tiles in the image.
transparency	optional, transparency value.

Returns

An integer handle to a sprite, 0 on error.

Description

Loads a sprite from the specified bitmap and adds it to the window. Width and height specify the dimensions of one image tile and frames specifies the number of tiles in the image. The optional key parameter sets the transparency color of the sprite.

Examples

```
DEF myWindow:WINDOW
DEF spr:INT
WINDOW myWindow,0,0,640,480,@NOAUTODRAW|@SIZE,0,"DXSprite",mainwindow


CREATESCREEN(myWindow,640,480)


'load the sprite
spr =  DXSPRITE(myWindow,"C:\Program Files\Ionic Wind\Creative
BASIC\samples\mouth.bmp",40,36,3,RGB(87,87,87))


run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END


mainwindow:
SELECT @CLASS
    CASE @IDDXUPDATE
        GOSUB update
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

```
SUB update
    'clear screen
    DXFILL myWindow,RGB(0,0,0)
    'draw sprite
    DXDRAWALLSPRITES myWindow
    'show the changes
    DXFLIP myWindow
RETURN
```

Loads in a sprite from C:\Program Files\Ionic Wind\Creative BASIC\samples\mouth.bmp, please change this path as required.

See Also

DXREMOVESPRITE

ELLIPSE

Usage

```
ELLIPSE winhandle, left, top, width, height {,bordercolor{,fillcolor}}
```

Parameters

`winhandle`	user defined WINDOW or DIALOG variable.
`left`	the distance between the left side of the window and the left side of the ellipse.
`top`	the distance between the top of the window and the top of the ellipse.
`width`	the width of the ellipse.
`height`	the height of the ellipse.
`bordercolor`	the color of the ellipse border. The RGB function can be used for this.
`fillcolor`	the color of the ellipse. The RGB function can be used for this.

Returns

Nothing.

Description

Draws an ellipse in the specified window, bound by the rectangle specified by left, top, width and height. If colors `bordercolor` and `fillcolor` are not specified the ellipse will be drawn in the current foreground color. Optional color `bordercolor` specifies the outline of the ellipse and `fillcolor` specifies the fill color.

Examples

```
DEF myWindow:WINDOW
DEF loopcount, left, width, run : INT
left=150
width=300
WINDOW myWindow,0,0,640,500,@SIZE|@MINBOX|@MAXBOX,0,"Ellipse Sample",main
CONTROL myWindow,"B,Close Window,0,0,200,20,0,1"
ELLIPSE myWindow,30,30,100,150,RGB(150,150,150),RGB(0,255,255)

for loopcount = 1 to 10
    ELLIPSE myWindow,left,left,width,width-20,RGB(255,0,0),RGB(0,0,255)
    width = width - 30
    left = left + 15
next loopcount



run = 1
WAITUNTIL run = 0
```

```
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
    SELECT @CONTROLID
        CASE 1
            run = 0
    endselect
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above program draws a simple ellipse, then uses a for loop to draw ever smaller ellipses in a pattern.

See Also

LINE; RECT

ELSE

Usage

```
IF ... {ELSE} ... ENDIF
```

Parameters

None.

Returns

Nothing.

Description

The ELSE statement is part of the IF statement. If the condition of the IF part of the statement is false, then all statements between the ELSE and ENDIF statements will be executed.

Examples

```
OPENCONSOLE
DEF myVariable AS INT
INPUT "Enter a number and press enter: ",myVariable
PRINT

IF myVariable > 10
    PRINT "Number was greater than 10"
ELSE
'number was not greater than 10
    IF myVariable < 10
        PRINT "Number was less than than 10"
    ELSE
    'since the number is not greater than, or less than 10 it must be 10
        PRINT "Number was 10"
    ENDIF
ENDIF
PRINT
```

```
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program reads in a number and then uses `IF` statements to find out if the number is greater than, less than, or equal to 10.

See Also

IF; ENDIF

ENABLECONTROL

Usage

```
ENABLECONTROL winhandle, controlID, enabled
```

Parameters

`winhandle`	user defined WINDOW or DIALOG variable.
`controlID`	the ID number of the control.
`enabled`	1 for enabled, 0 for disabled (grayed out).

Returns

Nothing.

Description

Enables or disables a control in the window or dialog. Use 0 to disable and 1 to enable.

Examples

```
DEF myWindow AS WINDOW
DEF enabled, run AS INT
WINDOW myWindow, 50, 50, 500, 500, @SIZE|@MINBOX|@MAXBOX, 0, "ENABLECONTROL Example",
main
CONTROL myWindow,"B,Close Window,10,0,100,20,0,1"
CONTROL myWindow,"B,sample button,10,100,100,20,0,2"
CONTROL myWindow,"B,enable / disable sample button,120,100,220,20,0,3"
enabled = 1

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
    SELECT @CONTROLID
        CASE 1
            run = 0
        CASE 3
            IF enabled = 1
                enabled = 0
            ELSE
                enabled = 1
```

```
            ENDIF
            ENABLECONTROL myWindow,2,enabled
    endselect
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above program creates 3 buttons - one to close the program, one which acts as a sample (so we can enable / disable it) and a third which enables / disables the sample button. When disabled, controls will appear greyed out, and the user of the program will not be able to do anything with these controls.

See Also

ENABLEMENU

Usage

```
ENABLEMENU winhandle, menuID, 0 | 1
```

Parameters

`winhandle`	user defined WINDOW variable.
`menuID`	the ID number of the menu.
`enabled`	1 for enabled, 0 for disabled (grayed out).

Returns

Nothing

Description

Enables or disables a menu in the window or dialog. Use 0 to disable and 1 to enable.

Examples

```
DEF myWindow AS WINDOW
DEF enabled, run AS INT
WINDOW myWindow, 50, 50, 500, 500, @SIZE|@MINBOX|@MAXBOX, 0, "ENABLECONTROL Example",
main
CONTROL myWindow,"B,Close Window,10,0,100,20,0,1"
CONTROL myWindow,"B,enable / disable sample button,120,100,220,20,0,2"
MENU myWindow,"T,&Test,0,0"
ADDMENUITEM myWindow,0,"Some Menu option",0,3
enabled = 1

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
    SELECT @CONTROLID
        CASE 1
            run = 0
        CASE 2
```

```
            IF enabled = 1
                enabled = 0
            ELSE
                enabled = 1
            ENDIF
            ENABLEMENU myWindow,0,enabled
    endselect
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above program creates 2 buttons and a "Test" menu, one button to close the program, and a second which enables / disables the Test menu. When disabled, menus will appear greyed out, and the user of the program will not be able to do anything with these menus. Disabling the menu will disable all items on the menu.

See Also

MENU; ADDMENUITEM

ENABLECONTROL

Usage

```
 ENABLEMENU winhandle, menuID, 0 | 1
```

Parameters

`winhandle`	user defined WINDOW or DIALOG variable.
`menuID`	the ID number of the menu.
`enabled`	1 for enabled, 0 for disabled (grayed out).

Returns

Nothing

Description

Enables or disables a menu item in the window or dialog. Use 0 to disable and 1 to enable.

Examples

```
DEF myWindow AS WINDOW
DEF enabled AS INT
WINDOW myWindow, 50, 50, 500, 500, @SIZE|@MINBOX|@MAXBOX, 0, "ENABLECONTROL Example",
main
CONTROL myWindow,"B,Close Window,10,0,100,20,0,1"
CONTROL myWindow,"B,enable / disable sample button,120,100,220,20,0,2"
MENU myWindow,"T,&Test,0,0"
ADDMENUITEM myWindow,0,"Some Menu option",0,3
enabled = 1

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
```

```
    SELECT @CONTROLID
        CASE 1
            run = 0
        CASE 2
            IF enabled = 1
                enabled = 0
            ELSE
                enabled = 1
            ENDIF
            ENABLEMENUITEM myWindow,3,enabled
    endselect
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above program creates 2 buttons and a "Test" menu, one button to close the program, one and a second which enables / disables the menu item on the Test menu. When disabled, menu items will appear greyed out, and the user of the program will not be able to do anything with these menu items. Disabling the menu item will only disable all items on the menu.

ENABLETABS

Usage

```
ENABLETABS winhandle, enable
```

Parameters

`winhandle`	user defined WINDOW or DIALOG variable.
`enable`	1 for enabled, 0 for disabled

Returns

Nothing

Description

If enable = 1 then the window's message loop allows tabbing between controls and using a default button. To disable this set enable = 0. Note that when enabled a window will not receive @IDCHAR messages

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT
WINDOW myWindow, 50, 50, 500, 500, @SIZE|@MINBOX|@MAXBOX, 0, "ENABLECONTROL Example",
main
CONTROL myWindow,"B,Close Window,10,0,100,20,@TABSTOP,1"
CONTROL myWindow,"E,,120,100,220,20,@TABSTOP,2"
CONTROL myWindow,"E,,120,130,220,20,@TABSTOP,3"
CONTROL myWindow,"B,Enable Tabbing,120,160,220,20,@TABSTOP,4"

'set the first edit control to have focus
SETFOCUS myWindow, 2

'move to a specified place on the window ready to print some text
MOVE myWindow,0,200
```

```
PRINT myWindow,"If you try pressing the tab key, nothing will happen initially."
MOVE myWindow,0,220
PRINT myWindow,"Click on the Enable Tabbing button and pressing tab"
MOVE myWindow,0,240
PRINT myWindow,"will move you between controls."

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS

        CASE @IDCONTROL
            SELECT @CONTROLID
                CASE 1
                    run = 0
                CASE 4
                    ENABLETABS myWindow,1
                    ENABLECONTROL myWindow,4,0
                    SETFOCUS myWindow, 2
            ENDSELECT

        CASE @IDCLOSEWINDOW
            run = 0

    ENDSELECT
RETURN
```

The above program creates some controls on a window. If you click on the enable tabbing button, then you will be able to move between the controls by pressing the tab button.

END

Usage

```
END
```

Parameters

None.

Returns

Nothing.

Description

Ends your program. Closes all windows, dialogs and open files. Clears all variables

Examples

```
OPENCONSOLE
PRINT "Press any key to close this program"

DO
```

```
UNTIL INKEY$ <>""
CLOSECONSOLE
END
```

The above program simply prints some text and when you press a key, the `END` command is activated, closing the program

ENDIF

Usage

```
IF ... {ELSE} ... ENDIF
```

Parameters

None.

Returns

Nothing

Description

The `ENDIF` statement will end an `IF` statement.

Examples

```
OPENCONSOLE
DEF myVariable AS INT
INPUT "Enter a number and press enter: ",myVariable
PRINT

IF myVariable > 10
    PRINT "Number was greater than 10"
ELSE
'number was not greater than 10
    IF myVariable < 10
        PRINT "Number was less than than 10"
    ELSE
    'since the number is not greater than, or less than 10 it must be 10
        PRINT "Number was 10"
    ENDIF
ENDIF
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program reads in a number and then uses `IF` statements to find out if the number is greater than, less than, or equal to 10.

See Also

IF; ELSE

ENDPAGE

Usage

```
 ENDPAGE printer
```

Parameters

`printer` a reference to a printer opened with the `OPENPRINTER` command.

Returns

Nothing.

Description

Ends the currently printed page and starts the next. This normally ejects the page from the printer (form feed). `printer` must be a validly open printer handle received from the `OPENPRINTER` function.

Examples

```
OPENCONSOLE
DEF myPrinter : INT

myPrinter = OPENPRINTER (GETDEFAULTPRINTER, "Test Document", "TEXT")
IF myPrinter
    ENDPAGE myPrinter
    CLOSEPRINTER myPrinter
ELSE
    PRINT "Unable to open the default printer."
ENDIF

CLOSEPRINTER myPrinter
CLOSECONSOLE
END
```

The above program will open the default printer, and send a command to the printer to eject the current page.

See Also

OPENPRINTER()

ENDSELECT

Usage

```
SELECT ... CASE ... {DEFAULT} ... ENDSELECT
```

Parameters

None.

Returns

Nothing.

Description

Ends a `SELECT` statement.

Examples

```
OPENCONSOLE
DEF Choice$: STRING
PRINT "Press some keys, Q to quit"
Choice$ = INKEY$

DO
    DO
        Choice$ = INKEY$
    UNTIL Choice$ <> ""

    SELECT Choice$
```

```
        CASE "A"
        CASE "a"
            PRINT "You pressed A!!"
        CASE "Z"
        CASE "z"
            PRINT "Z is my favorite letter!"
        CASE "Q"
        CASE "q"
            CLOSECONSOLE
            END
        DEFAULT
            PRINT "You pressed the letter ",Choice$
    ENDSELECT

UNTIL Choice$ ="q" | Choice$ ="Q"

CLOSECONSOLE
END
```

The above program simply checks which letter you pressed and prints that out to the screen. The ENDSELECT statement closes the opening SELECT statement.

See Also

SELECT

ENDTYPE

Usage

```
TYPE ... ENDTYPE
```

Parameters

None.

Returns

Nothing.

Description

Ends a TYPE statement

Examples

```
OPENCONSOLE
'create a new Creative BASIC variable type
TYPE contactdetails
   DEF Name:STRING
   DEF Age:INT
   DEF Phone[20]:ISTRING
ENDTYPE
'declare a variable of type contactdetails
DEF contact:contactdetails

'fill in the details of your contact into the variable
contact.Name = "Joe Smith"
contact.Age = 35
contact.Phone = "555-555-1212"
```

```
'Print the contact's details out
PRINT "Contact's Name:  ",contact.Name
PRINT "Contact's Age:   ",contact.Age
PRINT "Contact's Phone: ",contact.Phone
PRINT
PRINT "Press a key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program defines a new variable type (contactdetails) and uses a variable of this type (contact) to store and print a telephone contact's details.

See Also

TYPE

ENDWHILE

Usage

```
WHILE condition ... ENDWHILE
```

Parameters

None.

Returns

Nothing.

Description

The `ENDWHILE` statement marks the end of the corresponding `WHILE` statement. The condition in a `WHILE` loop is always tested before the loop content is executed, meaning that a while loop can execute zero or more times. A `DO` loop on the other hand, must be executed at least once.

Examples

```
OPENCONSOLE
DEF loopcount:INT
loopcount = 10

WHILE loopcount > 0
    PRINT "loopcount variable is set to : ",loopcount
    loopcount = loopcount - 1
ENDWHILE
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program uses a `WHILE` loop to print out the value of a variable, until the variable is less than 1.

See Also

WHILE

EOF()

Usage

```
EOF(myFile)
```

Parameters

`myFile` a handle to an open file.

Returns

Returns TRUE if the End Of File marker has been reached

Description

File must have been previously opened with the `OPENFILE` function.

Examples

```
DEF myFile:FILE
DEF fileText:STRING

OPENFILE(myFile,"C:\CREATIVE BASICTEST.TXT","W")
WRITE myFile,"Test string."
CLOSEFILE myFile

OPENFILE(myFile,"C:\CREATIVE BASICTEST.TXT","R")

WHILE EOF(myFile) = 0
    print eof(myFile)
    READ(myFile, fileText)
    PRINT fileText
ENDWHILE

CLOSEFILE myFile
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

A file is created and three strings are written to it. A `WHILE...DOWHILE` loop then iterates through the file until the end of the file (`EOF`) is reached.

See Also

OPENFILE; CLOSEFILE

EXP()

Usage

```
EXP(n)
```

Parameters

`n` any numeric expression.

Returns

A number - the exponential value of the numeric expression `n`.

Description

Returns the exponential value of the numeric expression `n`.

Examples

```
OPENCONSOLE
PRINT "EXP(5.6) = ", EXP(5.6)
```

```
PRINT
PRINT "Press any key to end this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program returns the value 270.43.

FILEREQUEST()

Usage

```
FILEREQUEST( title, parent, Save/Open{,filter}{,ext}{,flags}{,initial directory} )
```

Parameters

title	the text to appear in the dialogs title bar.
parent	the name of the WINDOW variable used to store the window. Use 0 if you are using a console application.
save / open	0 for a "Save As" dialog, 1 for a "File Open" dialog.
filter	an optional string containing ordered pairs. Each element of the string is separated with an '|' character and ending with two | |. The filter tells windows what kind of files are shown in the dialog. Example: Filter$ = "BITMAP files|*.BMP|GIF files|*.GIF||"
ext	the default extension used if the user does not type one. It should not contain a '.'
flags	to allow users to select multiple files supply @MULTISELECT for the flags parameter.
initial directory	the initial directory. For example "c:\" will begin in the root of the c: drive.

Returns

A string containing the fully qualified pathname to the user-selected file.

Description

Opens the standard file dialog and returns a string containing the fully qualified pathname to the user-selected file. For a 'Save as' dialog Save/Open should be 0. For an 'Open' type dialog Save/Open should be 1.

Examples

```
REM Define a buffer to hold the returned filenames.
REM Define a buffer to hold the returned filenames.
DEF filenames[10000]:ISTRING
DEF filter,filetemp:STRING
DEF pos:int
filter = "All Files (*.*)|*.*|Text Files (*.txt)|*.txt||"

filenames = FILEREQUEST("Select Multiple Files",0,1,filter,"txt",@MULTISELECT, "C:\")
DO
    pos = INSTR(filenames,"|")
    IF(pos)
        filetemp = LEFT$(filenames,pos-1)
        filenames = MID$(filenames,pos+1)
        REM do something with the file in filetemp
    ENDIF
```

```
UNTIL pos = 0
END
```

The above program opens a file open dialog, and uses the returned string (for multipe files) to open each file. The code does not edit the opened files. If you wanted to edit the files, you would dd appropriate code at the second REM statement.

FINDCLOSE()

Usage

```
FINDCLOSE(directory)
```

Parameters

`directory` a handle to the directory specified by the `FINDOPEN` command.

Returns

Nothing.

Description

Closes a directory opened with `FINDOPEN`.

Examples

```
OPENCONSOLE
DEF dir:INT
DEF filename,myDir:STRING
INPUT "Enter a valid directory eg C:\ ", myDir
myDir = myDir + "*.*"
dir = FINDOPEN(myDir)
IF(dir)
    PRINT
    PRINT "The contents of ",myDir," are as follows:"
    DO
        filename = FINDNEXT(dir)
        PRINT filename
    UNTIL filename = ""
    FINDCLOSE dir
ELSE
    PRINT "That was not a valid directory!!"
ENDIF

PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""

CLOSECONSOLE
END
```

The above example will ask for a directory, and print out the names of all the files in that directory, similar to the DOS prompt command DIR.

See Also

`FINDOPEN, FINDNEXT`

FINDNEXT()

Usage

```
FINDNEXT(directory)
```

Parameters

`directory` a handle to the directory specified by the `FINDOPEN` command.

Returns

The next filename in the chosen directory, returns an empty string ("") when all filenames have been returned.

Description

Retrieves the next filename in a directory. Used in a loop to retrieve all files.

Examples

```
OPENCONSOLE
DEF dir:INT
DEF filename,myDir:STRING
INPUT "Enter a valid directory eg C:\ ", myDir
myDir = myDir + "*.*"
dir = FINDOPEN(myDir)
IF(dir)
    PRINT
    PRINT "The contents of ",myDir," are as follows:"
    DO
        filename = FINDNEXT(dir)
        PRINT filename
    UNTIL filename = ""
    FINDCLOSE dir
ELSE
    PRINT "That was not a valid directory!!"
ENDIF

PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""

CLOSECONSOLE
END
```

The above example will ask for a directory, and print out the names of all the files in that directory, similar to the DOS prompt command DIR.

See Also

`FINDOPEN, FINDCLOSE`

FINDOPEN()

Usage

```
FINDOPEN(dir)
```

Parameters

`dir` a string the contains the full path to the directory plus any wildcard symbols for file matching. Example: "c:\windows*.txt".

Returns

Returns 0 if directory cannot be opened. An integer value is returned if it can be opened.

Description

To read all of the file names in a directory use the FINDOPEN function to get a handle to the directory first. Once a directory is opened the names of the files can be obtained with the FINDNEXT function.

Examples

```
OPENCONSOLE
DEF dir:INT
DEF filename,myDir:STRING
INPUT "Enter a valid directory eg C:\ ", myDir
myDir = myDir + "*.*"
dir = FINDOPEN(myDir)
IF(dir)
    PRINT
    PRINT "The contents of ",myDir," are as follows:"
    DO
        filename = FINDNEXT(dir)
        PRINT filename
    UNTIL filename = ""
    FINDCLOSE dir
ELSE
    PRINT "That was not a valid directory!!"
ENDIF

PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""

CLOSECONSOLE
END
```

The above example will ask for a directory, and print out the names of all the files in that directory, similar to the DOS prompt command DIR.

See Also

FINDNEXT, FINDCLOSE

FLOODFILL

Usage

```
FLOODFILL winhandle,x,y,color
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
x	the x position to begin the floodfill at.
y	the y position to begin the floodfill at.
color	the color to be used.

Returns

Nothing.

Description

Fills an area containing the point x,y with the specified color. Filling continues outward until a color other than the one specified is encountered. This can be used to fill a whole window with color, or if you specify a point in the middle of a shape (CIRCLE, ELLIPSE, RECT) it will color the shape only (the border being a different color, so the floodfill stops).

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,500,500,@SIZE|@MINBOX|@MAXBOX,0,"Flood Fill",main
CONTROL myWindow,"B,Close Window, 4,100,100,20,0,2"
CONTROL myWindow,"B,Floodfill circle, 4,130,100,20,0,3"
CONTROL myWindow,"B,Floodfill window, 4,160,100,20,0,4"
CIRCLE myWindow,250,75,50,RGB(0,0,255),RGB(255,255,255)

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 2
                run = 0
            CASE 3
                FLOODFILL myWindow, 255,80,RGB(255,0,0)
            CASE 4
                FLOODFILL myWindow, 0,0,RGB(0,255,0)
        ENDSELECT
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

The above program creates a window with some buttons and a circle in it. One button fills the circle with red (just the circle) the other fills the window with green (not the circle). The floodfill command will flood an area with color until it meets an area of different color. In the above program this area of different color is provided by the outline of the circle, so both the window and circle can be floodfilled without affecting the other.

FLOOR()

Usage

```
FLOOR(n)
```

Parameters

`n` a number.

Returns

The smallest integer less than or equal to `n`.

Description

Returns the floor of the specified number or expression. The floor of a number is the closest integer that is less than or equal to the number.

Examples

```
OPENCONSOLE
PRINT "FLOOR(5.5) = ",FLOOR(5.5)
PRINT "Press any key to end this program"
```

```
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above code returns 5.

See Also

CEIL

FONTREQUEST

Usage

```
FONTREQUEST(winhandle,size,weight,flags,color{,name})
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
size	an INT variable representing the size (in points) of the font.
weight	an INT variable representing the weight of the font, ranges from 0 to 1000 with 700 being standard for bold fonts and 400 for normal fonts.
flags	an INT variable representing flags, can be a combination of @SFITALIC, @SFUNDERLINE, or @SFSTRIKEOUT for italicized, underlined, and strikeout fonts.
color	an INT variable representing the color of the font.
name	an INT variable representing the initial font to be selected.

Returns

The name of the chosen font. FONTREQUEST returns an empty string if the user cancels the dialog. NB - this function also changes the variables passed as size, weight, flags and color so that they store the users choice for each of these properties.

Description

The FONTREQUEST function opens the standard system font dialog. The functions returns the name of the font and sets four variables with the attributes of the requested font. The variable parameters must be of type INT, and if values are set for these, they will be the initial values for the item in the system font dialog.

Examples

```
REM The following variables must be type int, and can be set to a default value
REM however these are changed by the function for later use.
DEF myWindow:WINDOW
WINDOW myWindow,0,0,500,500,@SIZE|@MINBOX|@MAXBOX,0,"Controls Sample",main
CONTROL myWindow,"B,Close Window, 4,100,100,20,0,2"
DEF size,weight,flags,col: INT
DEF fontname: STRING
fontname = FONTREQUEST (myWindow,size,weight,flags,col)
IF fontname <> ""
SETFONT myWindow,fontname,size,weight,flags
FRONTPEN myWindow,col
PRINT myWIndow,"And this is the result!!"
ENDIF

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
```

```
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 2
                run = 0
        ENDSELECT
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

Opens the FONTREQUEST dialog and allows the user to select the font properties before displaying a message to the screen.

See Also

SETFONT

FOR

Usage

```
 FOR loopvar = startno TO endno #stepno

----- OR -----



FOR loopvar = startno TO endno STEP stepno
```

Parameters

loopvar	a variable to hold the counter.
startno	the starting number.
endno	the ending number.
stepno	the amount to increment each loop, can be negative (decrement).

Returns

Nothing. The variable loopvar will be changed by the for loop.

Description

Begins a FOR NEXT loop. Step, a positive number, can be used to set an increment value (or negative for a decrement by value). If step is left out, the loop increments by 1 each loop. This type of loop is best suited for situtions when you know which values you want to use, for example setting the ID's of 10 controls. It would not be suitable for finding the names of every file in a directory, since you don't know how many files there may be.

Examples

```
REM ----- Example 1 -----
OPENCONSOLE
DEF loopcounter: INT
PRINT "Counting the numbers from 1 to 10"
FOR loopcounter = 1 to 10
    PRINT loopcounter
NEXT loopcounter
PRINT
```

```
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END

REM ----- Example 2 -----
OPENCONSOLE
DEF loopcounter: INT
PRINT "Counting the even numbers from 10 down to 1"
FOR loopcounter = 10 to 1 STEP -2
    PRINT loopcounter
NEXT loopcounter
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Example 1 simply prints out the numbers 1 to 10. Note how there is no `STEP` part to the loop. When this isn't present, Creative BASIC automatically increments the `loopvar` by 1.

Example 2 uses the `STEP` part to print out all the even numbers between 1 and 10, starting at 10 and working backward through 8,6,4,2 etc.

See Also

`NEXT; STEP`

FREELIB

Usage

```
FREELIB dllname
```

Parameters

`dllname` the name of the DLL to be unloaded.

Returns

Nothing.

Description

Unloads a DLL previously loaded with a `DECLARE` statement. Useful when a DLL is only needed for a short time.

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"ALIAS Sample",main
DECLARE "User32", MessageBoxA (wnd:window,text:string,title:string,flags:int),int
CENTERWINDOW myWindow
MessageBoxA(myWindow, "This messagebox has been called using the ALIAS command",
"Information", 0)
REM DLL no longer needed by the program so unload it...
FREELIB "User32"
```

```
PRINT myWindow,"The DLL was no longer needed, so it was unloaded with the FREELIB
command"


run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END


SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above program loads a messagebox DLL, displays a messagebox, then unloads the DLL since it will not be used by the program again.

See Also

DECLARE, ALIAS, CALL

FREEMEM

Usage

```
FREEMEM memory
```

Parameters

`memory` a variable defined as type memory and allocated with the `ALLOCMEM` statement.

Returns

Nothing.

Description

Free a previously allocated block of memory.

Examples

```
OPENCONSOLE
DEF buffer: MEMORY
DEF myNumber: INT
ALLOCMEM buffer, 100, LEN(myNumber)
PRINT "I have just allocated enough memory for 100 items"
FREEMEM buffer
PRINT "I have just freed the allocated memory"
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above example declares the variable buffer to be of type memory, myNumber to be an integer, then allocates enough memory to store 100 items the same size as the variable myNumber. It then frees up the allocated memory using the `FREEMEM` command.

See Also

ALLOCMEM

FRONTPEN

Usage

```
FRONTPEN winhandle, color
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
color	the value for the color you want. This is easily set by using the RGB function (see example).

Returns
Nothing.
Description
Sets the foreground color for all drawing operations in window to color.
Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Demo",main
PRINT myWindow,"Test Text before frontpen used  "
FRONTPEN myWindow, RGB(255,0,0)
PRINT myWindow,"Test Text after frontpen used"

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above code will set the frontpen color for myWindow equal to light grey.
See Also
RGB(), BACKPEN

GET

Usage

```
 GET file, record, var
```

Parameters

file	The name of the WINDOW variable used to store the window.
record	The position of the record you want to retrieve from the file.
var	The variable you want to store the retrieved information into.

Returns
The selected record's contents will be stored in the variable passed to the function.
Description
Gets one record from the random access binary file file and stores the result in var. record must be greater than zero and the file must have been opened by OPENFILE. var can be any built-in or user defined variable type. Use EOF to determine when end of file has been reached.
Examples

```
OPENCONSOLE
TYPE HighScoreTopTen
     DEF Score:INT
     DEF Time:INT
     DEF pname[20]:ISTRING
```

```
ENDTYPE
DEF loopcount : INT
DEF TopTen:HighScoreTopTen
DEF myfile:BFILE
IF(OPENFILE(myfile,"C:\HIGHSCORES.DAT","W") = 0)
    'Generate some data for the file
    FOR loopcount = 1 to 10
    TopTen.Score = loopcount * 10
    TopTen.Time = loopcount * 1.5
    TopTen.pname = "Player" + STR$(loopcount)
    PUT myfile,loopcount,TopTen
    NEXT loopcount
ELSE
    PRINT "Error opening the file"
ENDIF
GET myfile,8,TopTen
PRINT "The 8th record in the file is:"
PRINT
PRINT "Player Name  :",TopTen.pname
PRINT "Player Score :",TopTen.Score
PRINT "Player Time  :",TopTen.Time
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program writes some generated records into a file, then uses the get command to print out record number 8 to the screen.

See Also

PUT, OPENFILE, FOR, TYPE

GETBITMAPSIZE

Usage

```
GETBITMAPSIZE(image, width, height)
```

Parameters

image	The handle of an image loaded with the LOADIMAGE function.
width	The width of the loaded image. Must be an INT variable.
height	The height of the loaded image. Must be an INT variable.

Returns

Assigns the width and height to the variables passed to the function. Both variables must be of type INT.

Description

Returns the size, in pixels, of a bitmap loaded with the LOADIMAGE statement. width and height must be of type INT.

Examples

```
DEF myWindow : WINDOW
DEF imageWidth, imageHeight, myImage : INT
DEF filename : STRING
filename = filerequest("Select an image",0,1,"Bitmaps|*.bmp||","txt",0, "C:\")
myImage = LOADIMAGE (filename, @IMGBITMAP)
GETBITMAPSIZE(myImage, imageWidth, imageHeight)
WINDOW myWindow,0,0,imageWidth,imageHeight + 100,@SIZE|@MINBOX|@MAXBOX,0,"demo",main
CONTROL myWindow,"B,close window,4,4,100,20,0,1"
SHOWIMAGE myWindow, myImage, @IMGBITMAP, 0, 30, imageHeight, imageWidth
MOVE myWindow, 0, imageHeight + 30
PRINT myWindow, "Image Width : ", imageHeight, ", height : ", imageHeight

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 1
                run = 0
        ENDSELECT
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above program uses several image functions to dynamically design a window to contain a user selected image. The window is made big enough to contain the image, and some text printing the width and height of the image.

See Also

LOADIMAGE

GETCAPTION()

Usage

```
GETCAPTION(winhandle)
```

Parameters

`winhandle` the name of the `WINDOW`, or `DIALOG` variable used to store the window.

Returns

The caption of the window specified.

Description

Returns a string containing the caption text of the window.

Examples

```
DEF myWindow : WINDOW
WINDOW myWIndow, 50, 50, 500, 150, @SIZE|@MINBOX|@MAXBOX, 0, "GETCAPTION demo", main
```

```
CONTROL myWindow,"B,Close Window, 4,0,100,20,0,2"
CONTROL myWindow,"B,Show the caption of this window,4,60,300,20,0,3"

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
CASE @IDCONTROL
        SELECT @CONTROLID
        CASE 2
            run = 0
        CASE 3
            MESSAGEBOX myWindow,"The caption of this window is:" + CHR$(13) +
GETCAPTION(myWIndow),"Window Caption Information"
        ENDSELECT
CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

The above program displayes a message box showing the caption of the window when the button is clicked.

See Also

SETCAPTION

GETCARETPOSITION

Usage

```
GETCARETPOSITION winhandle, xpos, ypos
```

Parameters

winhandle	user defined WINDOW variable.
xpos	The x position of the system caret in the window. Must be an INT variable.
ypos	The y position of the system caret in the window. Must be an INT variable.

Returns

The position of the caret in the window is assigned to the variables passed to the function.

Description

Returns the current caret position in the window. xpos and ypos must be of type INT. The caret is the flashing vertical bar used when typing text into a textbox. This function is mainly used to gain information that can be used to manipulate text.

Examples

```
DEF myWindow : WINDOW
WINDOW myWindow,0,0,640,200,@SIZE|@MINBOX|@MAXBOX,0,"GETCARETPOSITION demo",main
DEF myYPos, myXPos : INT
CONTROL myWindow,"B,close window,4,4,100,20,0,1"
CONTROL myWindow,"B,display the caret position,4,34,200,20,0,2"

run = 1
```

```
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
    CASE @IDCREATE
        CONTROL myWindow,"E,,4,60,200,100,@CTEDITMULTI|@CTEDITRETURN|@VSCROLL,3"
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 1
                run = 0
            CASE 2
                GETCARETPOSITION myWindow, myXPos, myYPos
                MESSAGEBOX myWindow,"The caret is at position " + STR$(myXPos) +", " +
STR$(myYPos),"caret position"
        ENDSELECT
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above program allows you to type into the textbox. At any time you can click on the button to find out what the position of the caret is.

See Also

LOCATE; MOVE

GETCLIENTSIZE

Usage

```
GETCLIENTSIZE winhandle, left, top, width, height
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
left	the position of the window or dialog from the left. Must be passed an INT variable.
top	the position of the window or dialog from the lefttop. Must be passed an INT variable.
width	the width of the window or dialog. Must be passed an INT variable.
height	the height of the window or dialog. Must be passed an INT variable.

Returns

the variables passed to left, top, width and height are assigned the dimensions of the window.

Description

This function returns the position and dimension of he chosen window's client area. The client area of a window is the area of the window, minus any space taken up by title bars, scrollbars, statusbars etc.

Examples

```
DEF myWindow:WINDOW
DEF myLeft, myTop, myWidth, myHeight :INT
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Demo",main
CONTROL myWindow,"B,Close Window, 4,0,100,20,0,2"
CONTROL myWindow,"B,Show the client area of this window,4,60,300,20,0,3"

run = 1
```

```
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 2
                run = 0
            CASE 3
                GETCLIENTSIZE myWindow, myLeft, myTop, myWidth, myHeight
                MESSAGEBOX myWindow,"Left = " + STR$(myLeft) + CHR$(13) +"Top = " +
STR$(myTop) + CHR$(13) +"Width = " + STR$(myWidth) + CHR$(13) +"Height = " +
STR$(myHeight) + CHR$(13),"Dimensions of this window"
        ENDSELECT
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

By clicking on the button on the form, a message will display the dimensions of the client area of the window. You can see how this changes by moving, or resizing the window and clicking on the button.

See Also

GETSIZE

GETCONTROLTEXT

Usage

```
GETCONTROLTEXT winhandle, controlID
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
controlID	the ID of the control you want to get the text from.

Returns

The text of the control specified by controlID.

Description

Returns a string containing the text of the control specified by controlID. Can be used for controls located in windows or dialogs.

Examples

```
DEF myWindow : WINDOW
WINDOW myWindow,0,0,640,200,@SIZE|@MINBOX|@MAXBOX,0,"GETCONTROLTEXT demo",main
DEF myYPos, myXPos, run : INT
CONTROL myWindow,"B,close window,4,4,100,20,0,1"
CONTROL myWindow,"B,display the text of this control,4,34,200,20,0,2"

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END
```

```
SUB main
    SELECT @CLASS
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 1
                run = 0
            CASE 2
                MESSAGEBOX myWindow,"The text on control 2 is: " + CHR$(13) +
GETCONTROLTEXT(myWindow, 2),"Control text position"
        ENDSELECT
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above program uses the GETCONTROLTEXT function toreturn the text on the button when you click on it.

See Also

SETCONTROLTEXT

GETDEFAULTPRINTER

Usage

```
GETDEFAULTPRINTER
```

Parameters

None.

Returns

The name of the default printer.

Description

Returns a string containing the name of the default printer. A default printer must be set up on the user's computer. This name can be used in subsequent calls to the OPENPRINTER function.

Examples

```
DEF hPrt: INT
DEF data: STRING
DEF name: STRING
DEF pagefrom,pageto,copies,collate: INT
pagefrom = 1
pageto = 1
copies = 1
hPrt = OPENPRINTER (GETDEFAULTPRINTER,"Test Document","TEXT")
IF (hPrt)
    data = "This is a test of printing"
    data = data + chr$(13)
    data = data + "This is line 2"
    WRITEPRINTER hPrt,data
    CLOSEPRINTER hPrt
ENDIF
END
```

The above code opens the defaultprinter, sends some text to the printer for printing, then closes the printer.

See Also

OPENPRINTER,CLOSEPRINTER

GETDXVERSION

Usage

```
GETDXVERSION
```

Parameters

None.

Returns

The major version number of DirectX installed on the system.

Description

Returns the major version number of DirectX installed on the system. Returns 0 if DirectX is not available. By checking that this number is greater than or equal to the version your program needs, you can advise the user to update their version of DirectX. This will mean that the user will not come across any errors when running your program.

Examples

```
OPENCONSOLE
PRINT "The version of DirectX installed on this computer is : ",GETDXVERSION
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above example prints out the version of DirectX installed on the computer.

GETHDC()

Usage

```
GETHDC(winhandle)
```

Parameters

`winhandle` user defined `WINDOW` variable.

Returns

A handle to the Device Context of the window.

Description

For advanced users. Returns a handle to the Device Context of a window. Handle can be used in subsequent calls to windows GDI functions.

Examples

```
DEF myWindow:WINDOW
DEF myHandle : INT
WINDOW myWindow,0,0,300,260,@SIZE|@MINBOX|@MAXBOX,0,"HDC Sample",main
CONTROL myWindow,"B,Close Window,0,0,200,20,0,1"
myHandle = GETHDC(myWIndow)

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END
```

```
SUB main
SELECT @CLASS
    CASE @IDCONTROL
    SELECT @CONTROLID
        CASE 1
            run = 0
    endselect
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above example assigns the Device Context handle of the window to a variable. It is not possible to print this value out.

See Also

RELEASEHDC

GETKEYSTATE()

Usage

```
GETKEYSTATE(keycode)
```

Parameters

keycode	the virtual keycode of the key.

Returns

The status of the specified virtual key.

Description

Returns the status of the specified virtual key. Return value is > 0 if key is pressed.

NOTE: GETKEYSTATE will not work for a console only program under Windows 95/98 or ME.

Examples

```
OPENCONSOLE
PRINT "This program will print a message when the enter key is pressed"
DO
UNTIL GETKEYSTATE(08) > 0
PRINT "You just pressed the enter key"
PRINT
PRINT "Press any key to close this program"
DO UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

DOES NOT WORK

See Also

INKEY

GETPIXEL()

Usage

```
GETPIXEL(winhandle, xpos, ypos)
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
width	the width of the loaded image. Must be an INT variable.

height	the height of the loaded image. Must be an INT variable.

Returns
The color of the pixel at point xpos, ypos.

Description

Returns the color of the pixel at position xpos, ypos in the window. The number returned is the decimal representation of the color. You can use the returned value in any of the functions that require a color

For advanced users: To convert this into RGB values, you must convert this number to hex by using the HEX command, then split the string into 3 strings of 2 characters, and finally convert each string into decimal.

Examples

```
DEF myWindow:WINDOW
DEF loopcount, left, width : INT
left=150
width=300
WINDOW myWindow,0,0,300,260,@SIZE|@MINBOX|@MAXBOX,0,"Get Pixel",main
CONTROL myWindow,"B,Close Window,0,0,200,20,0,1"
ELLIPSE myWindow,30,30,100,150,RGB(150,150,150),RGB(0,255,255)
MOVE myWindow, 0, 200
PRINT myWindow, "The color of pixel 200,0 is :", GETPIXEL(myWindow,200,0)
MOVE myWindow, 0, 215
PRINT myWindow, "The color of pixel 65,105 is :", GETPIXEL(myWindow,65,105)
run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
    SELECT @CONTROLID
        CASE 1
            run = 0
    endselect
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above program draws an ellipse, and chooses one pixel not in the ellipse, and one pixel in the ellipse then prints the color value of these.

See Also
HEX, RGB, ELLIPSE

GETPOSITION

Usage

```
GETPOSITION winhandle, xpos, ypos
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
xpos	the x position of the current drawing position.
ypos	the y position of the current drawing position.

Returns
Assigns the x and y position of the current drawing position into the variables passed as xpos and ypos.

Description

Retrieves the current drawing position in the window. xpos and ypos must be of type INT. The current drawing position is set with the MOVE statement, graphics operations, and text PRINTing.

Examples

```
DEF myWindow : WINDOW
WINDOW myWindow,0,0,640,200,@SIZE|@MINBOX|@MAXBOX,0,"demo",main
DEF myYPos, myXPos : INT
CONTROL myWindow,"B,close window,4,4,100,20,0,1"
MOVE myWindow, 0,50
GETPOSITION myWindow, myXPos, myYPos
PRINT myWindow,"The drawing position is currently at : ",myXPos,",",myYPos
MOVE myWindow, 10,80
GETPOSITION myWindow, myXPos, myYPos
PRINT myWindow,"The drawing position is currently at : ",myXPos,",",myYPos
MOVE myWindow, 20,110
GETPOSITION myWindow, myXPos, myYPos
PRINT myWindow,"The drawing position is currently at : ",myXPos,",",myYPos

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 1
                run = 0
        ENDSELECT
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above program uses the GETPOSITION function to print out the current drawing position. Any time you use PRINT, MOVE or graphics functions this will change, hence the printed values in the program above are correct until they are printed to the window, at which point the drawing position will have changed.

See Also

PRINT, MOVE

GETSCREENSIZE

Usage

```
GETSCREENSIZE width,height
```

Parameters

width	the width of the screen. Must be passed an INT variable.
height	the height of the screen. Must be passed an INT variable.

Returns

The width and height of the screen are assigned to the variables passed to the function.

Description

Retrieves the system screen size and stores the results in varW and varH. The variables must be of type INT. This function is very useful when designing a window or dialog that will be the size of the screen, no matter what screen resolution the user's computer is set to.

Examples

```
OPENCONSOLE
DEF screenWidth, screenHeight : INT
GETSCREENSIZE screenWidth, screenHeight
PRINT "The dimensions of the screen are:"
PRINT "Width -",STR$(screenWidth)," Height -",STR$(screenHeight)
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program uses the `GETSCREENSIZE` function to print out the dimensions of the screen.

GETSCROLLPOS()

Usage

```
GETSCROLLPOS(winhandle,scrollID)
```

Parameters

`winhandle`	user defined `WINDOW` or `DIALOG` variable.
`scrollID`	if ID = -1 then returns the scroll position of the windows horizontal scrollbar. If ID = -2 then returns the scroll position of the windows vertical scrollbar. Any other ID value is a user-defined scrollbar.

Returns

The slider position of a scrollbar.

Description

Returns the slider position of a scrollbar. If ID = -1 then returns the scroll position of the windows horizontal scrollbar. If ID = -2 then returns the scroll position of the windows vertical scrollbar. Any other ID value is a user-defined scrollbar.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX | @HSCROLL | @VSCROLL, 0,
"Window 1 Example", main

SETSCROLLRANGE myWindow,-1,0,300
SETSCROLLRANGE myWindow,-2,0,300

SETSCROLLPOS myWindow,-1,50
PRINT myWindow,"Scroll Position: ",GETSCROLLPOS(myWindow,-1)

run = 1
WAITUNTIL run = 0
```

```
END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
RETURN
```

Sets up a window with two scroll bars and moves the horizontal one along by 50, then uses GETSCROLLPOS to display the value in the window.

See Also

SETSCROLLPOS

GETSCROLLRANGE

Usage

```
GETSCROLLRANGE winhandle, scrollID, minvalue, maxvalue
```

Parameters

winhandle	user defined WINDOW variable.
scrollID	if ID = -1 then stores the range of the windows horizontal scrollbar. If ID = -2 then stores the range of the windows vertical scrollbar. ID must be a scrollbar control.
minvalue	the minimum scroll value of the chosen scrollbar. Must be an INT variable.
maxvalue	the maximum scroll value of the chosen scrollbar. Must be an INT variable.

Returns

Assigns the minimum and maximum scrollvalues of the chosen scrollbar to the variables passed to the function.

Description

Stores the scrollbars range into the variables specified by varMin and varMax. The variables must be of type INT. If ID = -1 then stores the range of the windows horizontal scrollbar. If ID = -2 then stores the range of the windows vertical scrollbar. ID must be a scrollbar control.

Examples

```
DEF myWindow AS WINDOW
DEF run,minpos,maxpos AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX | @HSCROLL | @VSCROLL, 0,
"Window 1 Example", main

SETSCROLLRANGE myWindow,-1,0,300
SETSCROLLRANGE myWindow,-2,0,300
SETSCROLLPOS myWindow,-1,50
GETSCROLLRANGE(myWindow,-1,minpos,maxpos)
PRINT myWindow,"Scroll Range: ", minpos," & ", maxpos

run = 1
WAITUNTIL run = 0

END
SUB main
IF @CLASS = @IDCLOSEWINDOW
```

```
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
RETURN
```

Uses GETSCROLLRANGE to store the min and max scroll range of myWindow in the variables minpos and maxpos and then displays them on the screen.

See Also

SETSCROLLRANGE

GETSELECTED()

Usage

```
GETSELECTED(winhandle, controlID)
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
controlID	the ID number of the control.

Returns

The zero-based index of the currently selected item in a list or combo box.

Description

Returns the zero-based index of the currently selected item in the list box of a combo or single selection list box. Returns -1 if no item is selected. ID must be a list box or combo box.

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"INSERTMENU Demo",main
CONTROL myWindow,"M,,250,100,300,100,@CTCOMBODROPDOWN,1"
ADDSTRING myWindow,1,"Option 1"
ADDSTRING myWindow,1,"Option 2"
ADDSTRING myWindow,1,"Option 3"
CONTROL myWindow,"B,Close Window, 4,0,100,20,0,2"
CONTROL myWindow,"B,Add an option to the combo box, 4,30,250,20,0,3"
CONTROL myWindow,"B,Find the selected item, 4,60,300,20,0,4"

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 2
                run = 0
            CASE 3
                INSERTSTRING myWindow, 1, 0, "This has just been added"
            CASE 4
```

```
                MESSAGEBOX myWindow,STR$(GETSELECTED(myWindow,1)),"The currently
selected option is:"
        ENDSELECT
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

The above program contains two buttons, one will add an option to a combo box, and the other will display the ID of the currently selected option in the combobox. It will return -1 if there is no current selection.

See Also

SETSELECTED

GETSIZE

Usage

```
GETSIZE winhandle, left, top, width, height{, controlID}
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
left	the position of the window, dialog or control from the left. Must be passed an INT variable.
top	the position of the window, dialog or control from the lefttop. Must be passed an INT variable.
width	the width of the window, dialog or control. Must be passed an INT variable.
height	the height of the window, dialog or control. Must be passed an INT variable.
controlID	an optional ID for a visible control.

Returns

The left and top poitions, width and height of the window, dialog or control are assigned to the variables passed to the function.

Description

Stores the size of the window, dialog or control in the variables. controlID is optional and must be the identifier of a valid, visible control. Size includes the titilebar, borders and scrollbars if a window.

Examples

```
DEF myWindow:WINDOW
DEF myLeft, myTop, myWidth, myHeight :INT
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Demo",main
CONTROL myWindow,"B,Close Window, 4,0,100,20,0,2"
CONTROL myWindow,"B,Show the dimesnsion of this window,4,60,300,20,0,3"

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 2
                run = 0
            CASE 3
```

```
                GETSIZE myWindow, myLeft, myTop, myWidth, myHeight
                MESSAGEBOX myWindow,"Left = " + STR$(myLeft) + CHR$(13) +"Top = " +
STR$(myTop) + CHR$(13) +"Width = " + STR$(myWidth) + CHR$(13) +"Height = " +
STR$(myHeight) + CHR$(13),"Dimensions of this window"
        ENDSELECT
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

By clicking on the button on the form, a message will display the dimensions of the window. You can see how this changes by moving, or resizing the window and clicking on the button.

See Also

SETSIZE; GETCLIENTSIZE

GETSTARTPATH

Usage

```
GETSTARTPATH
```

Parameters

None.

Returns

The full path of the directory which contains the current program executable.

Description

Used to get the full path of the directory which contains the current program executable. Useful when you want to access files relative to the current directory.

Examples

```
OPENCONSOLE
PRINT "The current directory is:"
PRINT GETSTARTPATH
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program simply prints out the current directory.

GETSTATE()

Usage

```
 GETSTATE (winhandle, ID)
```

Parameters

`winhandle` user defined `WINDOW` or `DIALOG` variable.

`controlID` the ID number of the control.

Returns

Returns 1 if the control is checked, 0 if it is not checked.

Description

Returns the checked/unchecked state of a radio button or checkbox control.

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"INSERTMENU Demo",main
CONTROL myWIndow,"C,,400,30,10,10,0,1"
CONTROL myWindow,"B,Close Window, 4,0,100,20,0,2"
CONTROL myWindow,"B,Find out if the radio button is checked, 4,60,300,20,0,3"

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 2
                run = 0
            CASE 3
                MESSAGEBOX myWindow,"check status = " + STR$(GETSTATE(myWindow,1)) +
CHR$(13) +"0 = not checked" + CHR$(13) +"1 = checked","Check status of radio button"
        ENDSELECT
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

The above program has a button and a checkbox on it. Pressing the button will display a messagebox with the current ststus of the checkbox on it. Clicking on the checkbox will change this status, so that when the button is clicked again, it displays this altered status.

See Also

SETSTATE

GETSTRING()

Usage

```
GETSTRING (winhandle, controlID, pos)
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
controlID	the ID number of the control.
pos	the position of the option you want to find the text of.

Returns

Returns the string at pos in a list box or combobox.

Description

Returns the string at pos in a list box or combobox. pos is a zero based index.

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"GetString Demo",main
CONTROL myWindow,"M,,250,100,300,100,@CTCOMBODROPDOWN,1"
ADDSTRING myWindow,1,"Option 1"
```

```
ADDSTRING myWindow,1,"Option 2"
ADDSTRING myWindow,1,"Option 3"
CONTROL myWindow,"B,Close Window, 4,0,100,20,0,2"
CONTROL myWindow,"B,Add an option to the combo box, 4,30,250,20,0,3"
CONTROL myWindow,"B,Find the text in position 0, 4,60,300,20,0,4"

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 2
                run = 0
            CASE 3
                INSERTSTRING myWindow, 1, 0, "This has just been added"
            CASE 4
                MESSAGEBOX myWindow,GETSTRING(myWindow,1,0),"Option 0's text is"
        ENDSELECT
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

The above program contains two buttons, one will add an option to a combo box, and the other will display the text displayed in position 0 of the combobox.

See Also

MESSAGEBOX, ADDSTRING, INSERTSTRING

GETSTRINGCOUNT()

Usage

```
GETSTRINGCOUNT (winhandle, controlID)
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
controlID	the ID number of the control.

Returns

The number of test strings in a list box or combo box control.

Description

Returns the number of test strings in a list box or combo box control. This function returns the number of options in the combo box, but when referencing these, remember that the ID of the first option is 0, so the last option will have an ID of one less than the value returned to this function.

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Getstringcount Demo",main
CONTROL myWindow,"M,,250,100,300,100,@CTCOMBODROPDOWN,1"
```

```
ADDSTRING myWindow,1,"Option 1"
ADDSTRING myWindow,1,"Option 2"
ADDSTRING myWindow,1,"Option 3"
CONTROL myWindow,"B,Close Window, 4,0,100,20,0,2"
CONTROL myWindow,"B,Add an option to the combo box, 4,30,250,20,0,3"
CONTROL myWindow,"B,how many options are in the combo box, 4,60,300,20,0,4"

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 2
                run = 0
            CASE 3
                INSERTSTRING myWindow, 1, 0, "This has just been added"
            CASE 4
                MESSAGEBOX myWindow,STR$(GETSTRINGCOUNT(myWindow,1)),"Number of
options in the combo box"
        ENDSELECT
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

The above program contains two buttons, one will add an option to a combo box, and the other will display a message box which will show the number of options in the combo box.

See Also

MESSAGEBOX, ADDSTRING, INSERTSTRING

GETEXTSIZE

Usage

```
GETEXTSIZE winhandle, string, width, height
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
string	the text you want to measure.
width	the width the text will be in the window. Must pass an INT variable.
height	the height the text will be in the window. Must pass an INT variable.

Returns

The variables passed in to height and width will be assigned the width and height the text will be in the window.

Description

Returns the size of a string in pixels when printed to the window. width and height must be of type INT. The size of the string is useful for determining line positions.

Examples

```
DEF myWindow:WINDOW
DEF textHeight, textWidth : INT
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"GetTextSize Demo",main
CONTROL myWindow,"B,Close this window,250,100,300,20,0,1"
'position at 0,0 ready for printing there
MOVE myWindow, 0, 0
'print out some text to the window
PRINT myWindow, "This is the text"
'get the
GETTEXTSIZE myWindow, "This is the text", textWidth, textHeight
'move to the end of the sentence you just printed
MOVE myWindow, textWidth, 0
'We can now print on the window at the end of the above text,
'making it look like one string, instead of two strings
PRINT myWindow, " I want to print to this window"

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
        CASE @IDCONTROL
        SELECT @CONTROLID
            CASE 1
                run = 0
        ENDSELECT
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above program uses the GETTEXTSIZE function to add some text to the end of a string that was already printed on the window. This is useful for printing to the end of an existing string, or for printing more than one line. Being able to get the width and height of a piece of text will remove the guesswork from lining up text.

See Also

MOVE

GETUSERDATA

Usage

```
GETUSERDATA winhandle, variable
```

Parameters

winhandle	user defined WINDOW or DIALOG variable.
variable	the variable you want to store the data in .

Returns

The contents of the requested variable.

Description

Retrieves data previously set with SETUSERDATA. Variable may be of type INT, UINT or POINTER, depending on the type of data that was stored.

Examples

```
DEF myWindow : WINDOW
DEF myInt : INT
myInt = 567890
WINDOW myWindow,0,0,640,200,@SIZE|@MINBOX|@MAXBOX,0,"GETUSERDATA demo",main
SETUSERDATA myWindow, myInt :'temporarily store the value of myInt in the window
myInt = 0 :'reset myInt to 0
GETUSERDATA myWIndow, myInt :'store the windows userdata in myint
PRINT myWindow,myInt

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

The above program stores an int value in the window temporarily, and retrieves the information from the window into variable, which gets printed out.

See Also

SETUSERDATA

GOSUB

Usage

```
GOSUB {$}label
```

Parameters

`label` the name of the subroutine you want to use.

Returns

Nothing. Will return a value if the subroutine returns a value.

Description

Jumps to a subroutine defined by the SUB statement. When the subroutine ends by executing a RETURN statement then execution continues at the next line. If `label` is preceded by the $ character then it is treated as a variable. This allows creation of dynamic GOSUB statements. All variables defined from within a subroutine are local and will be destroyed when the RETURN statement is executed. A subroutine may be called by just using its name, without the GOSUB statement. Subroutines declared with the DECLARE statement must be called with the subroutine name.

Examples

```
DEF Jumpstring:STRING
Jumpstring = "MySubroutine"
REM This is the formal type of gosub
GOSUB MySubroutine
```

```
REM This is the dynamic version
GOSUB $Jumpstring
REM This is the implied version
MySubroutine
END

SUB MySubroutine
PRINT "I am here!"
RETURN
```

The above program shows the main ways of calling a subroutine.

See Also

DECLARE, SUB, RETURN

GOTO

Usage

```
GOTO {$}label
```

Parameters

label	a predefined label.

Returns

Nothing.

Description

Jumps to a predefined label statement. Interchangeable with JUMP. As with JUMP, be aware that usage of this command is not recommended, as it makes your code less readable, since the reader must continually jump around following your code. If the label is preceded with $ then it is treated as a variable, therefore allowing the creation of dynamic GOTO statements.

Examples

```
OPENCONSOLE
DEF myString:STRING
LABEL start
INPUT "Enter a letter, q to quit: ",myString
PRINT "You pressed the letter : ",myString
PRINT
IF myString <> "q"
    GOTO start
ELSE
    CLOSECONSOLE
ENDIF
END
```

The above program will ask for a letter to be pressed, and will print out which letter you pressed until you pres the letter q, at which point the program will quit.

See Also

LABEL, JUMP

HEX$()

Usage

```
HEX$(n)
```

Parameters

n	a number.

Returns

A string representing the hexadecimal notation of parameter `n`.

Description

Converts the numeric expression `n` to a string representing the hexadecimal (base 16) notation version of that number.

Examples

```
OPENCONSOLE
DEF myNumber : INT
INPUT "Enter a number:",myNumber
PRINT "HEX$(",STR(myNumber)," = ",HEX$(myNumber)
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program asks for a number and prints out the hexadecimal version of that number.

IF

Usage

```
IF ... {ELSE} ... ENDIF
```

Parameters

None.

Returns

Nothing.

Description

If the condition of the `IF` part of the statement is false, then all statements between the `ELSE` and `ENDIF` statements will be executed.

Examples

```
OPENCONSOLE
DEF myVariable AS INT
INPUT "Enter a number and press enter: ",myVariable
PRINT

IF myVariable > 10
    PRINT "Number was greater than 10"
ELSE
'number was not greater than 10
    IF myVariable < 10
        PRINT "Number was less than than 10"
    ELSE
    'since the number is not greater than, or less than 10 it must be 10
        PRINT "Number was 10"
    ENDIF
ENDIF
PRINT
PRINT "Press any key to close this program"
```

```
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program reads in a number and then uses `IF` statements to find out if the number is greater than, less than or equal to 10.

See Also

`ELSE`; `ENDIF`

INKEY$

Usage

```
INKEY$ {( raw)}
```

Parameters

`raw` lets `INKEY$` return virtual keycodes.

Returns

A single character if a key hs been pressed, an empty string ("") if no key was pressed.

Description

Returns a single character from the keyboard in console mode. `OPENCONSOLE` must have been executed before using `INKEY$`. Return value is a character and can be stored in a string, integer or char variable. Returns and empty string if no key has been pressed. Optional raw parameter lets `INKEY$` return virtual keycodes.

Examples

```
OPENCONSOLE
DEF Choice$: STRING
PRINT "Press a key, Q to quit"
Choice$ = INKEY$
DO
    DO
    Choice$ = INKEY$
    UNTIL Choice$ <> ""

    SELECT Choice$
    CASE "A"
    CASE "a"
        PRINT "You pressed A!!"
    CASE "Z"
    CASE "z"
        PRINT "Z is my favorite letter!"
    CASE "Q"
    CASE "q"
        CLOSECONSOLE
        END
    DEFAULT
        PRINT "You pressed the letter ",Choice$
    ENDSELECT
UNTIL Choice$ ="q" | Choice$ ="Q"
```

Accepts user input with INKEY then passes the result into a SELECT CASE.

See Also

GETKEYSTATE

INPUT

Usage

```
INPUT {string,}variable
```

Parameters

string	the text to be displayed. This usually prompts the user for input.
variable	the variable used to store the users input.

Returns

The data input by the user is stored in the parameter variable.

Description

Gets data from the console window. OPENCONSOLE must have been executed prior to using the input statement. The optional string is a prompt. The statement will wait until the user has entered the proper data, and pressed return. Valid variable types are CHAR, INT, FLOAT, DOUBLE and STRING.

Examples

```
OPENCONSOLE
DEF myName : STRING
DEF myAge : INT
INPUT "Please enter your name :",myName
PRINT
INPUT "Please enter your age :",myAge
PRINT "Hello ", myName,"!!! On your next birthday, you will be ", STR$(myAge + 1)
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program asks for the users name and ageusing the INPUT command, it then prints out a sentence with that name in it, and prints out what age they will be on their next birthday.

INSERTMENU

Usage

```
INSERTMENU winhandle, pos, definition{, definition.}
```

Parameters

winhandle	a user defined WINDOW variable.
position	the position of the menu, 0 being the leftmost menu.
definition	a string containing:"TYPE,LABEL,FLAGS,ID" where: TYPE is a single character and can be 'T' for title, 'I' for item or 'S' for submenu. LABEL is the text to be displayed on the menu. Use "-" to display a menu separator. FLAGS are 2 predefined attributes for a new menu item.

@MENUDISABLE - The menu item will be grayed out and not selectable.

@MENUCHECK - Shows a checkmark next to the menu item.

If you don't want to use either of the above, you can use the number 0 to create a standard menu item.

`ID` is the id number of the menu.

Returns
Nothing.
Description
Inserts a menu into the window or dialog at the position specified by `position`. Each `definition` is a string constant that contains the data necessary for each menu-item. The format of the string constant is "`TYPE,LABEL,FLAGS,ID`". `TYPE` is a single character and can be 'T' for title, 'I' for item or 'S' for submenu. For a menu separator use '-' as the label. For a `MDI` window `INSERTMENU` should be used first instead of the `MENU` statement since a menu already exists. For all other types of window, you must use the `menu` command first.
Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"INSERTMENU Demo",main
MENU myWindow, "T, File, 0, 0" ,  "I, New, 0, 1" ,  "I, Quit, 0 ,2"
INSERTMENU myWindow,1, "T,Edit,0,0" , "I,Cut,0,3" ,  "I,Copy,0,4"


run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

Setups a window with the menu File, and inserts another menu called Edit.
See Also
`TYPE`; `MENU`; `ADDMENUITEM`

INSERTSTRING

Usage

```
 INSERTSTRING winhandle, ID, position, text
```

Parameters

`winhandle`	a user defined `window` or `dialog` variable.
`ID`	the ID of the listbox or combo box.
`position`	position to insert string at.
`text`	the string to insert

Returns
Nothing.
Description
Inserts a string into a combo or listbox control at position. All other strings are moved down by one position.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX | @HSCROLL | @VSCROLL, 0,
"InsertString Example", main

CONTROL myWindow,"L,ListBox1,115,29,72,121,0x50800140,1"
CONTROL myWindow,"B,Insert,20,72,70,20,0x50000000,2"

ADDSTRING myWindow,1,"1"
ADDSTRING myWindow,1,"2"
ADDSTRING myWindow,1,"3"

SETSELECTED myWindow,1,1

run = 1
WAITUNTIL run = 0

END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
IF (@CLASS = @IDCONTROL)
  INSERTSTRING myWindow, 1, 2, "Inserted String!"
ENDIF
RETURN
```

Once the button is pressed a string is inserted into the listbox (ID 1) at position 2.

See Also

ADDSTRING; INSERTSTRING

INSTR

Usage

```
INSTR ({start,} string1, string2)
```

Parameters

start	optional, INT.
string1	a string.
string2	a string.

Returns

An integer detailing the start position of the second string within the first.

Description

Used to locate the start of a string (a sub-string) within another string.

Examples

```
string1 = "The quick brown fox"
string2 = "quick"
```

```
position = INSTR(string1, string2)
print "'quick' is located at charcter ", position, " in 'string1'."
```

The program searches for the start position of the string "quick" within string1. In this example, the value 5 ("quick" starts at the 5th character) is returned.

INT

Usage

```
INT(number)
```

Parameters

`number` a floating point number.

Returns

Returns an INT.

Description

`INT` converts a floating point number to an integer. The supplied value is always rounded down.

Examples

```
PRINT INT(4.9)
```

The example displays the result '4'.

ISARRAY

Usage

```
ISARRAY(expression)
```

Parameters

`expression` a pointer or a string to be checked.

Returns

Returns an INT.

Description

`ISARRAY` returns 1 if the variable referenced by a pointer is an array. `ISARRAY` can also be used with variables types other than `POINTER`. Since a string is actually an array of characters, `ISARRAY` always returns 1 for a string variable.

Examples

```
myString = "ABC"
IF ISARRAY(myString)
   PRINT "That's an array!"
ELSE
   PRINT "That isn't an array."
ENDIF
```

The example checks to see if myString is an array. As all strings are arrays, this test should always prove positive. Note that even a single character string is classed as an array.

ISSELECTED

Usage

```
 ISSELECTED ( winhandle, ID, pos )
```

Parameters

`winhandle` a user defined `WINDOW` or `DIALOG` variable.

`ID` a control ID.

`pos` position in the control to check.

Returns

True or False.

Description

Returns 1 if the string if 'pos' is selected in a list or combo box.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX | @HSCROLL | @VSCROLL, 0,
"ISSELECTED Example", main

CONTROL myWindow,"L,ListBox1,115,29,72,121,0x50800140,1"

ADDSTRING myWindow,1,"0"
ADDSTRING myWindow,1,"1"
ADDSTRING myWindow,1,"2"

SETSELECTED myWindow,1,1

IF ISSELECTED(myWindow,1,0) = 1
    PRINT myWindow,"Item at position 0 is selected"
ENDIF

IF ISSELECTED(myWindow,1,1) = 1
    PRINT myWindow,"Item at position 1 is selected"
ENDIF

IF ISSELECTED(myWindow,1,2) = 1
    PRINT myWindow,"Item at position 2 is selected"
ENDIF

run = 1
WAITUNTIL run = 0

END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
RETURN
```

Checks the items in the listbox to see which item is selected.

See Also

SETSELECTED

JUMP

Usage

```
 JUMP label
```

Parameters

label a predefined label.

Returns

Nothing.

Description

Jumps to a predefined label statement. Interchangeable with GOTO. as with GOTO, be aware that usage of this command is not recommended, as it makes your code less readable, since the reader must continually jump around following your code.

Examples

```
OPENCONSOLE
DEF myString:STRING
LABEL start
INPUT "Enter a letter, q to quit: ",myString
PRINT "You pressed the letter : ",myString
PRINT
IF myString <> "q"
    JUMP start
ELSE
    CLOSECONSOLE
ENDIF
END
```

The above program will ask for a letter to be pressed, and will print out which letter you pressed until you pres the letter q, at which point the program will quit.

See Also

LABEL;GOTO

LABEL

Usage

```
LABEL name
```

Parameters

name simply the name you wish to call the label.

Returns

Nothing.

Description

Creates a label in your program that you can immediately jump to using GOTO or JUMP.

Examples

```
PRINT "The 2nd print statement is never run,"
GOTO skip
PRINT "THIS ISN'T DISPLAYED."
LABEL skip
PRINT "but the 3rd one is."
```

A line of text is printed and then program execution is jumped past the 2nd print statement.

See Also

GOTO; JUMP

LCASE$()

Usage

```
LCASE$(string)
```

Parameters

string a string.

Returns

A string.

Description

Returns the string parameter converted to all lowercase letters

Examples

```
DEF userString AS string
DEF newString AS string



OPENCONSOLE
userString = "Hello"

newString = LCASE$(userString)
PRINT newString

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Converts a string into lowercase.

See Also

UCASE

LEFT$()

Usage

```
LEFT$(string, characters)
```

Parameters

string	the original string to take the characters from.
characters	the number of characters to return from the string.

Returns

Returns the number of characters specified from the left hand side of the given string.

Description

Returns the number of characters specified from the left hand side of the given string.

Examples

```
OPENCONSOLE
name = "Dave Normal"
PRINT LEFT$(4, name)
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The example will only display the first four letters of the string, in this case, "Dave".

See Also

MID$;RIGHT$; LEN()

LEN()

Usage

```
LEN(variable | fileHandle)
```

Parameters

variable	a normal variable

`fileHandle` the handle of an open file

Returns

For a variable, returns the length in bytes of the variable (or the number of characters of a string).

For a file, returns the length of the data stored in the file.

Description

Used to determine the size of a variable in bytes (or the number of characters in a string). `LEN` is also used to return the size of any data stored in a file.

Examples

```
DEF name$ AS STRING
OPENCONSOLE
name$ = "John Sensible"
length = LEN(name$)
PRINT length
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

A string is stored in name$, and the size of the string is then stored in the "length" variable.

See Also

MID$() ; LEFT$() , RIGHT$()

LINE

Usage

```
LINE winhandle, x1, y1, x2, y2, colour
```

Parameters

`winhandle` a user defined `WINDOW` or `DIALOG` variable.
`x1` the horizontal starting point (in pixels) of the line.
`y1` the vertical starting point (in pixels) of the line.
`x2` the horizontal end point of the line.
`y2` the vertical end point of the line.
`color` the RGB value (color) of the line.

Returns

Nothing.

Description

Draws a straight line from the points x1, y1 to the points x2, y2 using the specified color.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window Example", main
LINE myWindow, 20, 20, 290, 180, RGB(255,0,0)

run = 1
WAITUNTIL run = 0

CLOSEWINDOW myWindow
END

SUB main
```

```
SELECT @CLASS
CASE @IDCLOSEWINDOW
run = 0
ENDSELECT
RETURN
```

The example opens a window and draws a single red line. The program then waits for the window to be closed.

See Also

RECT; CIRCLE; ELLIPSE

LOADIMAGE()

Usage

```
LOADIMAGE (filename | resource ID, type)
```

Parameters

filename	the name of a file to be loaded from disk.
resourceID	if specified instead of a filename, the image data will be taken from a resource compiled with the project.
type	the image type: @IMGBITMAP - BitMaP (*.bmp) @IMGICON - Icon (*.ico) @IMGCURSOR - Cursor (*.cur) @IMGEMF - Enhanced meta file (*.emf) @IMGSCALABLE - scalable bitmap, JPEG (*.jpg) or GIF (*.gif)

Returns

Returns a handle (INT) to the loaded image.

Description

Creative BASIC can load an image, icon or cursor with the LOADIMAGE function. LOADIMAGE returns a handle to the loaded image as an integer value. This value can then be passed to any of the functions that accepts a handle as a parameter.

Examples

```
DEF myWindow AS WINDOW
DEF pic,run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "LOADIMAGE Example",
main

pic = LOADIMAGE("C:\mypic.bmp",@IMGBITMAP)
SHOWIMAGE myWindow,pic,@IMGBITMAP,0,0,100,100

run = 1
```

```
WAITUNTIL run = 0
END

SUB main
   IF @CLASS = @IDCLOSEWINDOW
      CLOSEWINDOW myWindow
      run = 0
   ENDIF
RETURN
```

An image is loaded and given the handle "pic". The image is then displayed in the window using the SHOWIMAGE command.

See Also

SHOWIMAGE

LOADMENU()

Usage

```
LOADMENU(winhandle, resourceID)
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
resourceID	resource ID to load the menu from.

Returns

0 on failure, 1 on success.

Description

Loads a menu from resources and sets the menu bar for the dialog or window.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "LoadMenu Example",
main

LOADMENU(myWindow,22)

run=1
WAITUNTIL run=0

END

SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run=0
ENDIF
RETURN
```

After entering the above code you must click the resource menu item and add the following MENU resource as resource ID 22:

```
BEGIN
   POPUP "&File"
   BEGIN
      MENUITEM "&New\tCtrl+N", 2025
      MENUITEM "&Open...\tCtrl+O", 2026
      MENUITEM SEPARATOR
      MENUITEM "P&rint Setup...", 2027
      MENUITEM SEPARATOR
      MENUITEM "Recent File", 2028
      MENUITEM SEPARATOR
      MENUITEM "E&xit", 2029
   END
   POPUP "&Help"
   BEGIN
      MENUITEM "&About HelpExample...", 2030
   END
END
```

Once you have done this, you must compile an EXE and run the program from the EXE for the resource to work.

See Also

INSERTMENU; MENU

LOADRESOURCE()

Usage

```
success = LOADRESOURCE(resourceID, Type, Variable)
```

Parameters

resourceID	a numeric or string identifier to the resource.
type	a numeric or string type: @RESCURSOR @RESBITMAP @RESICON @RESMENU @RESDIALOG @RESSTRING @RESACCEL @RESDATA @RESMESSAGETABLE @RESGROUPCURSOR @RESGROUPICON @RESVERSION
variable	a copy of the resource, or an integer pointer to the resource.

Returns

0 on failiure, 1 on success.

Description

Loads a resource from the executable file and stores it in the variable specified. Variable may be of type STRING, ISTRING, MEMORY or INT.

If 'variable' is a STRING (or ISTRING) variable the contents of the resource will be copied into the string. Useful for embedded text resources.

If 'variable' is an INT variable then a handle to the locked resource is returned. Useful for API calls.

If 'variable' is a MEMORY variable then memory is allocated for the resource and resource copied into the memory. You can then use READMEM to read the resource data. LEN(variable) will return the length of the resource in this case. You must free the memory returned in this case with the FREEMEM statement. Only use a newly defined MEMORY variable or one that has been freed with FREEMEM. Using a MEMORY variable that has been allocated with ALLOCMEM will result in memory leaks in your program.

Examples

```
OPENCONSOLE

DEF out[20000]:ISTRING

IF(LOADRESOURCE(7777,"MYTYPE",out))
     PRINT out
ELSE
     PRINT "unable to load resource"
ENDIF

DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

This example requires you to add a resource (from the Resource / Add menu) as ID 7777, custom type called MYTYPE, and select "a .txt" file to load in. You must then compile the code to make an EXE for the resource example to work, as it loads the resource from the EXE file.

See Also

LOADMENU()

LOADTOOLBAR

Usage

```
success = LOADTOOLBAR ( window, handle | resID, barID, button_array[], style )
```

Parameters

Returns

returns 0 on error

Description

Loads and creates a tool bar control. Bitmap for the toolbar can be loaded from a file or directly from resources.

Examples

```
DEF hbitmap:UINT
DEF tbArray[8]:INT
DEF myWindow AS WINDOW
tbArray = 2,3,4,0,5,6,7,8
```

```
WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Toolbar Example", main

hbitmap = LOADIMAGE("C:\toolbar.bmp",@IMGBITMAP | @IMGMAPCOLORS)

REM this toolbar is loaded from a bitmap handle
IF LOADTOOLBAR(myWindow,hbitmap,998,tbArray,@TBTOP | @TBFROMHANDLE)
    CONTROLCMD myWindow,998,@TBENABLEBUTTON,3,0
ENDIF

run=1
WAITUNTIL run=0

END

SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run=0
ENDIF
RETURN
```

Loads in a toolbar using C:\toolbar.bmp - please change the path to your toolbar image as required.

See Also

LOADIMAGE

LOCATE

Usage

```
LOCATE y, x
```

Parameters

y	the row in the console at which to position the print cursor.
x	how many characters in to position the print cursor.

Returns

Nothing.

Description

This is used to position the cursor before using the `PRINT` command. Note that positioning is done using y, x and not x, y.

Examples

```
OPENCONSOLE
LOCATE 10, 5
PRINT "This text starts at the 10th row down, and the 5th character in."
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Locates the cursor at the 5th character on the 10th row, and displays some text.

See Also

PRINT

LOG()

Usage

```
LOG(n)
```

Parameters

`n` any numeric expression

Returns

A number - the natural logarithm of `n`.

Description

Returns the natural LOGarithm of the numeric expression `n`.

Examples

```
OPENCONSOLE
PRINT "LOG(4.2) = ", LOG(4.2)
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program returns 1.44.

See Also

LOG10

LOG10()

Usage

```
LOG(n)
```

Parameters

`n` any numeric expression

Returns

A number - the base 10 logarithm of `n`.

Description

Returns the base 10 LOGarithm of the numeric expression `n`.

Examples

```
OPENCONSOLE
PRINT "LOG10(7.4) = ", LOG10(7.4)
DO
UNTIL INKEY$ <> ""
END
```

The above program returns 0.87.

See Also

LOG

LTRIM$()

Usage

```
LTRIM$(string)
```

Parameters

`string` a string.

Returns

A string.

Description

Removes all leading white-space characters from a string. White-space characters include spaces and tabs.

Examples

```
OPENCONSOLE
myString = "     ABCD"
myString = LTRIM$(myString)
PRINT myString
CLOSECONSOLE
END
```

LTRIM$ is used to remove the leading spaces from myString.

See Also

RTRIM$

MENU

Usage

```
MENU winhandle, menuitem{, menuitem2,...}
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
menuitem	a string constant that contains the data necessary for each menu item
menuitem2	optional, a string constant that contains the data necessary for each menu item

Returns

Nothing.

Description

The format of the string constant is "TYPE,LABEL,FLAGS, ID". TYPE is a single character and can be 'T' for title, 'I' for item or 'S' for submenu. For a menu separator use '-' as the label. For an MDI window INSERTMENU should be used first instead of the MENU statement since a menu already exists.

Examples

```
MENU w,"T,&Test,0,0","I,&Minimize,0,6","I,Ma&ximize,0,7","I,&Restore,0,8"
```

Creates a menu called Test with the menu items Maximize, Minimize and Restore.

See Also

ADDMENUITEM; REMOVEMENUITEM

MESSAGEBOX

Usage

```
MESSAGEBOX window,text,caption {,flags}
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
text	string of text to be shown in the box.
caption	the caption of the messagebox.
flags	optional, controls the creation of the message box: @IDOK @IDCANCEL

Returns

Standard IDs: @IDOK and @IDCANCEL.

Description

Displays a message box in the window. Text and caption are strings that define what is shown in the box and caption. The optional flags parameter control the creation of the message box. MESSAGEBOX returns standard ID's of @IDOK and @IDCANCEL.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT
```

```
WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Messagebox Example",
main

CONTROL myWindow,"B,Button,114,72,70,20,0x50000000,1"
run=1
WAITUNTIL run=0

END

SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run=0
ENDIF

IF (@CLASS = @IDCONTROL)
    MESSAGEBOX myWindow,"Test Message","Msg Box!"
ENDIF
RETURN
```

Displays a test message using the message box once the button is pressed.

See Also

SENDMESSAGE

MID$()

Usage

```
Result$ = MID$(string, position {,count })
```

Parameters

string	a string to check.
position	position in the string to start from.
count	optional, number of characters from position to count.

Returns

a string.

Description

Extracts count number of characters staring at position from a string. If count is omitted all of the characters from position to the end of the string are returned.

Examples

```
DEF myString,Result AS STRING
myString = "Hello World"
OPENCONSOLE
Result = MID$(myString,7)
PRINT Result
END
```

Takes the string "Hello World" and from 7 characters across to the end strips out "World" with MID$ and prints it to the screen.

See Also

LEFT$; RIGHT$

MOVE

Usage

```
MOVE winhandle,x,y
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
x	on screen x coordinate to move to.
y	on screen y coordinate to move to.

Returns

Nothing.

Description

Moves the current pen position in the window to coordinates x,y.

Examples

```
MOVE window1,20,20
```

Moves the pen position to position 20,20 inside a window named window1.

See Also

FRONTPEN

NEW()

Usage

```
pointer = NEW(VariableType, length)
```

Parameters

pointer	a pointer to the newly created variable.
VariableType	the type of variable to create.
length	the size of the new variable - if greater than 1, then a single dimensional array is created.

Returns

The memory address of the variable.

Description

Creates a new dynamic variable. Size must be 1 or greater. Returns a pointer to the newly created variable.

Examples

```
OPENCONSOLE
DEF p : POINTER

'Create an INT in memory
p = NEW(INT, 1)
#p = 75
PRINT #p

'delete the INT
DELETE p

PRINT "Press any key to close"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

This defines `p` as a `POINTER`, and creates a new `INT` variable, assigning it to p. The value of 75 if placed in the address pointed to by `p` (using the `#` operator), and printed to the screen. The memory used by the variable is then freed using the `DELETE` command.

See Also

DELETE; TYPEOF

NEXT

Usage

```
NEXT variable
```

Parameters

`variable` the variable used as the loop counter in the corresponding `FOR` statement.

Returns

Nothing.

Description

Used to define the end of a `FOR...NEXT` loop.

Examples

```
FOR loop = 1 to 10
   PRINT loop
NEXT loop
```

The variable "loop" iterates through the values of 1 to 10, each value being printed to the screen.

See Also

`FOR`

NOT

Usage

```
NOT variable
```

Parameters

variable the value whose ones compliment number shall be returned

Returns

Returns the ones compliment of the number supplied.

Description

Take the supplied value and returns the ones compliment of that value.

Examples

```
DEF myInt AS INT
myInt = 42
PRINT myInt
PRINT NOT myInt
```

Displays the value of myInt and its one compliment value.

OPENCONSOLE

Usage

```
OPENCONSOLE
```

Parameters

None.

Returns

Nothing.

Description

Opens the system console for text based input and output. Be sure to call `CLOSECONSOLE` at the end of your program. Only one console exists in the system so only one can be open. The console is similar to an MSDOS window.

Examples

```
OPENCONSOLE
PRINT "This is an example of how to use OPENCONSOLE"
PRINT
PRINT "This window will stay open until you press a key"

DO
```

```
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Opens a console and displays some text. Pressing a key closes the console and ends the program.

See Also

CLOSECONSOLE

OPENFILE

Usage

```
OPENFILE filevar, filename, flags
```

Parameters

filevar	the variable defined by the user to handle the file (either a FILE or BFILE).
filename	the filename (including a path if necessary).
flags	either "R" (read from a file), "W" (write to a file), or "A" (append an existing file).

Returns

0 if the file was successfully opened.

Description

Prepares a file for read/write/append operations. If the file doesn't exist, then using the "W" parameter will ceate a new file.

Examples

```
OPENCONSOLE
DEF myfile: FILE
IF (OPENFILE (myfile,"C:\CREATIVE BASICTEST.TXT","W") = 0)
    WRITE myfile,"This is a test"
    CLOSEFILE myfile
    PRINT "File created successfully"
ELSE
    PRINT "File could not be created"
ENDIF
PRINT "Press Any Key To Close"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program attempts to create a file in the C:\ directory called CREATIVE BASICTEXT.TXT using the OPENFILE command with the "W" (write) parameter.

It writes a line of text to the file, then closes the file using the CLOSEFILE command.

See Also

CLOSEFILE

OPENPRINTER()

Usage

```
 OPENPRINTER(name, title, mode)
```

Parameters

name	this is a string returned by GETDEFAULTPRINTER or PRINTERDIALOG.
title	a string used to display the document name in the windows print spooler.
mode	either "TEXT" or "RAW" data mode.

Returns
Returns an integer handle to the printer specified or 0 if the printer could not be opened.
Description
Opens a printer connection.
Examples
```
DEF hPrt: INT
DEF data: STRING
DEF name: STRING
DEF pagefrom,pageto,copies,collate: INT
pagefrom = 1
pageto = 1
copies = 1
name = PRTDIALOG (0,pagefrom,pageto,copies,collate)
hPrt = OPENPRINTER (name,"Test Document","TEXT")
IF (hPrt)
    data = "This is a test of printing"
    data = data + chr$(13)
    data = data + "This is line 2"
    WRITEPRINTER hPrt,data
    CLOSEPRINTER hPrt
ENDIF
END
```

The example opens a printer, sends some text to the printer for printing, then closes the printer.
See Also
CLOSEPRINTER; WRITEPRINTER

PLAYWAVE
Usage
```
PLAYWAVE sound, flags
```
Parameters
sound	a WAV file or sound loaded into memory.
flags	play mode.

Returns
Nothing.
Description
Plays a WAV file specified by filename or previously loaded into memory. Valid values for flags are: @SNDASYNC, @SNDSYNC, @SNDLOOP, @SNDNOSTOP
Examples
```
OPENCONSOLE
PLAYWAVE "C:\Windows\Media\chimes.wav",0
PRINT "PRESS ANY KEY TO QUIT"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Plays the wav file located at C:\Windows\Media\chimes.wav, with flag 0. Please note, this .wav file may not exist on all computers, if not, please change the path to point to a wav file.

PRINT
Usage

```
PRINT {winhandle,}{expressionlist}
```

Parameters

winhandle	optional, window to print to (if none is specified prints to console).
expressionlist	optional, list of expressions or user defined variables, or literal strings seperated by commas.

Returns

Nothing.

Description

Outputs data to a window or the system console. If a window variable is omitted all output will be redirected to the console, which must have been opened with the OPENCONSOLE command. Expressions may be variables, functions or literals.

Examples

```
PRINT 2 + 2
PRINT "hello world"
```

Prints the result of the expression 2 + 2, and also prints a string literal.

See Also

LOCATE; INPUT

PRINTWINDOW

Usage

```
PRINTWINDOW window
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.

Returns

Nothing.

Description

Opens the standard printer dialog and sends the contents of the window to the selected printer. Output is properly scaled and sized for both portrait and landscape modes.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT
WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window Example", main
PRINT myWindow,"Hello, printer test page!"
run = 1
WAITUNTIL run = 0

PRINTWINDOW myWindow

CLOSEWINDOW myWindow
END
SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

Sends the window myWindow to the printer.

See Also

WRITEPRINTER

PRTDIALOG()

Usage

```
printername = PRTDIALOG(winhandle, pagefrom, pageto, copies, collate)
```

Parameters

winhandle	a user defined WINDOW variable.
pagefrom	page to print from.
pageto	page to print to.
copies	number of copies to print.
collate	an INT variable representing the collate options.

Returns

Returns the name of the selected printer, as a string.

Description

Opening the printer dialog allows the user to select the printer of their choice, pages to print, number of copies and whether you need to collate the copies.

Examples

```
DEF hPrt:INT
DEF data:STRING
DEF name:STRING
DEF pagefrom,pageto,copies,collate:INT
pagefrom = 1
pageto = 1
copies = 1
name = PRTDIALOG(0,pagefrom,pageto,copies,collate)
hPrt = OPENPRINTER(name,"Test Document","TEXT")
IF (hPrt)
    data = "This is a test of printing"
    data = data + chr$(13)
    data = data + "This is line 2"
    WRITEPRINTER hPrt,data
    CLOSEPRINTER hPrt
ENDIF

END
```

Returns the printer from the PRTDIALOG command then uses OPENPRINTER and sends it to printer.

See Also

PRINTWINDOW

PSET

Usage

```
 PSET winhandle, x, y {,color}
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
x	pixel x coordinate.
y	pixel y coordinate.
color	optional, the pixel color.

Returns

Nothing.

Description

Changes one pixel at position x,y in the window to the color specified or if color is omitted to the last color set by the `FRONTPEN` statement.

Examples

```
DEF myWindow AS WINDOW
DEF run,xcoord,ycoord,red,green,blue AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window Example", main
PRINT myWindow,"pset example"

FOR counter = 1 TO 500
    xcoord = RND(300)
    ycoord = RND(300)
    red = RND(255)
    green = RND(255)
    blue = RND(255)
    PSET myWindow,xcoord,ycoord,RGB(red,green,blue)
NEXT counter

run = 1
WAITUNTIL run = 0

CLOSEWINDOW myWindow
END
SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

Executes a loop, 500 `PSET`s at random coords on the window, with random colours.

See Also

`GETPIXEL`

PUT

Usage

```
PUT filehandle, position, var
```

Parameters

`filehandle`	file to access.
`position`	position in file.
`var`	variable to write to file.

Returns

Nothing.

Description

Writes one record to the random access binary file - filehandle. Record must be greater than 0. var can be any built-in or user defined variable type. filehandle must have been successfully opened for writing with the OPENFILE statement.

Examples

```
OPENCONSOLE
DEF myfile:BFILE
DEF mynumber:INT
DEF currentposition:INT
mynumber = 3000000
IF(OPENFILE(myfile,"C:\CREATIVE BASICDATA.DAT","W") = 0)
   PUT myfile,1,mynumber
   CLOSEFILE myfile
   PRINT "mynumber PUT to file"
ENDIF

PRINT "Press Any Key To Close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

PUTs mynumber into the file providing it does not already exist.

See Also

GET

RASTERMODE

Usage

```
RASTERMODE winhandle, mode
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
mode	the mode to set: @RMBLACK @RMWHITE @RMNOP @RMNOT @RMCOPYPEN @RMNOTCOPYPEN @RMMERGEPENNOT @RMMASKPENNOT @RMMERGENOTPEN @RMMASKNOTPEN @RMMERGEPEN @RMNOTMERGEPEN @RMMASKPEN @RMNOTMASKPEN @RMXORPEN @RMNOTXORPEN

Returns

Nothing.

Description

Sets the current raster mode for the window.

Examples

```
DEF w:WINDOW
```

```
WINDOW w,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Demo",main
CENTERWINDOW w
RASTERMODE w, @RMXORPEN
RECT w,20,20,50,100,RGB(0,0,255),RGB(0,255,0)
ELLIPSE w,20,20,100,50,RGB(255,0,0),RGB(255,255,0)
CIRCLE w,50,50,25,RGB(0,255,0),RGB(0,0,255)
run = 1

WAITUNTIL run = 0
CLOSEWINDOW w
END

SUB main
SELECT @CLASS
     CASE @IDCLOSEWINDOW
                run = 0
ENDSELECT
RETURN
```

Sets the window rastermode to @RMXORPEN and draws 3 shapes to the window.

See Also

FRONTPEN, BACKPEN

READ()

Usage

```
READ

 file, var, {var...}
```

Parameters

filehandle	a user defined file handle.
var	variable to store read data to.

Returns

0 on success.

Description

Reads data from a file opened with the OPENFILE command.

Examples

```
OPENCONSOLE
DEF myfile:FILE
DEF ln:STRING
IF(OPENFILE(myfile,"C:\CREATIVE BASICTEST.TXT","R") = 0)
   IF(READ(myfile,ln) = 0)
            PRINT ln
   ENDIF
   CLOSEFILE myfile
   PRINT "File read successfully"
ELSE
   PRINT "File could not be opened"
```

```
ENDIF
PRINT "Press Any Key To Close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Reads in data from C:\CREATIVE BASICTEST.TXT - be sure to create this file and place some text inside it.

See Also

WRITE; OPENFILE; CLOSEFILE

READMEM

Usage

```
READMEM memory, position, variable {,offset}
```

Parameters

memory	variable of type memory, allocated with the ALLOCMEM command.
position	must be greater than 0.
variable	any built in, or user-defined variable type.
offset	optional, where reading and writing will occur.

Returns

Nothing.

Description

Gets one record from memory and stores the result in variable. position must be greater than zero and memory must have been successfully allocated with ALLOCMEM or returned from a system message or function. variable can be any built-in or user defined variable type.

Examples

```
OPENCONSOLE
DEF num:INT
DEF buffer:MEMORY
DEF buffer2:MEMORY
TYPE mytype
    DEF name[40]:ISTRING
    DEF age:INT
    DEF money:FLOAT
ENDTYPE
DEF record:mytype
ALLOCMEM buffer,100,LEN(num)
ALLOCMEM buffer2,10,LEN(record)

record.name = "John Doe"
record.age = 32
record.money = 100.01

WRITEMEM buffer2,1,record
READMEM buffer2,1,record
PRINT record.name
PRINT record.age
PRINT record.money
FREEMEM buffer
```

```
FREEMEM buffer2

PRINT "Press any key to continue"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Writes the record to memory, then reads it from memory and displays it to the screen.

See Also

WRITEMEM; ALLOCMEM

RECT

Usage

```
RECT window, left, top, height, width {, border, fill}
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
left	the left position to draw to.
top	the top position to draw to.
width	the width of the rectangle.
height	the height of the rectangle.
border	optional, the border colour of the rectangle.
fill	optional, the fill colour of the rectangle.

Returns

Nothing.

Description

Draws a rectangle in the window. If a border color is specified then the rectangle is drawn in that color. If a fill color is specified then the rectangle is filled. If neither color is specified an unfilled rectangle is drawn using the last FRONTPEN color.

Examples

```
DEF w:WINDOW
WINDOW w,0,0,200,320,@SIZE|@MINBOX|@MAXBOX,0,"Lines",main
CENTERWINDOW w
RECT w,50,50,100,200,RGB(255,0,0),RGB(0,0,255)
run = 1

WAITUNTIL run = 0
CLOSEWINDOW w
END
SUB main
SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

Draws a rectangle to the window with a border and fill colour.

See Also

LINE; ELLIPSE; CIRCLE

RELEASEHDC

Usage

```
RELEASEHDC window, HDC
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
HDC	the HDC to be freed.

Returns

Nothing.

Description

For advanced users. Frees a previously acquired HDC. To avoid consuming resources you should release any HDCs before ending your program.

Examples

```
DEF myWindow AS WINDOW
DEF run,handle AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "HDC Example", main
handle = GetHDC myWindow
CONTROL myWindow, "B, Button, 114, 72, 70, 20, 0x50000000, 1"
run = 1
WAITUNTIL run = 0

END

SUB main
    IF @CLASS = @IDCLOSEWINDOW
        REM closes the window and sets w1 = 0
        CLOSEWINDOW myWindow
        run = 0
    ENDIF

    IF (@CLASS = @IDCONTROL)
        ReleaseHDC myWindow, handle
    ENDIF
RETURN
```

Creative BASIC allows using WIN32 (WINAPI) functions to draw in a window. In order to use any of the WIN32 graphics functions you must first obtain a handle to a device context (HDC). The handle returned is an integer value and is a valid HDC. The handle is released once the button is pressed.

See Also

GETHDC

REM

Usage

```
REM
```

Parameters

None.

Returns

Nothing.

Description

REMark. Any text after this statement is ignored. Used to place comments in your code.

Examples

```
OPENCONSOLE
```

```
REM This is a comment and will not be interpreted.
REM Neither will this.
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The lines beginning with REM are just comments and will be ignored by Creative BASIC. Instead of using REM, a single apostrophe can be used to comment a line instead.

REMOVEDIR

Usage

```
REMOVEDIR name
```

Parameters

name	an empty directory

Returns

0 on error.

Description

Removes the directory specified by name. The directory must be empty in order to remove it.

Examples

```
OPENCONSOLE
testdir = "C:\testdir\"
REMOVEDIR testdir
PRINT "FOLDER REMOVED"
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

For this example to work you need to create an empty folder on C:\ called "testdir".

See Also

CREATEDIR; DELETEFILE

REMOVEMENUITEM

Usage

```
REMOVEMENUITEM window | dialog, position, itemID
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
position	the zero-based index of the menu starting from the left.
itemID	the menu item to remove.

Returns

Nothing.

Description

Removes a menu item. If itemID = 0 then the entire menu specified by position is removed.

Examples

```
REM define a window variable
DEF w1:WINDOW
REM open the window
WINDOW w1,0,0,350,350,@MINBOX|@MAXBOX|@SIZE,0,"Simple Window",main
REM MENU w1,"T,Options,0,0","I,Close,0,1"
```

```
MENU w1, "T, File, 0, 0" ,  "I, Remove Me, 0, 1" ,  "I, Quit, 0 ,2"
REM print a message
PRINT w1,"Hello World"
REM when w1 = 0 the window has been closed
WAITUNTIL w1 = 0
END

REM every time there is a message for our window
REM the operating system will "GOSUB" here
SUB main
    IF @CLASS = @IDCLOSEWINDOW
        REM closes the window and sets w1 = 0
        CLOSEWINDOW w1
    ENDIF

    IF (@CLASS = @IDMENUPICK) & (@MENUNUM = 2)
        CLOSEWINDOW w1
    Else
        IF (@CLASS = @IDMENUPICK) & (@MENUNUM = 1)
            REM Removes the menu item at position 0,1.
            REMOVEMENUITEM w1, 0, 1
        ENDIF
    ENDIF
RETURN
```

Creates a menu item which deletes itself when clicked.

See Also

ADDMENUITEM; MENU

REPLACE$

Usage

```
REPLACE$ string, start, count, new
```

Parameters

string	string to use the replace on.
start	start point for replace.
count	how many characters to replace.
new	the string to replace with

Returns

Nothing.

Description

Replaces characters in one string with characters from another.

Examples

```
DEF s: string
OPENCONSOLE
s = "All good DOGs go to heaven"
REPLACE$ s,10,3,"dog"
PRINT s
DO
```

```
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Replaces 3 characters starting at the 10th character ("DOG") with "dog". "All good dogs go to heaven" will be printed.

See Also

MID$()

RETURN

Usage

```
RETURN {variable | expression}
```

Parameters

variable	optional variable to return.
expression	optional expression to return.

Returns

A user defined variable or expression.

Description

Returns from a subroutine called by GOSUB, dialog procedure or window procedure. Optionally returns the value of a variable or expression.

Examples

```
DECLARE calculate(x: FLOAT ,y: FLOAT )
OPENCONSOLE
PRINT calculate(25,75)
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END

SUB calculate(x,y)
    DEF ret: FLOAT
    ret = x + y
RETURN ret
```

RETURNs the variable ret which is then printed to the screen.

See Also

DECLARE; SUB; GOSUB

RGB()

Usage

```
RGB(r,g,b)
```

Parameters

r	red value (0-255).
g	green value (0-255).
b	blue value (0-255).

Returns

Nothing.

Description

Returns a color reference of the combined RGB value. Most windows statements expect a combined value. The values can range from 0 to 255 indicating the level (brightness) of the component.

Examples

```
DEF myWindow:WINDOW
```

```
WINDOW myWindow,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Demo",main
PRINT myWindow,"Test Text before frontpen used  "
FRONTPEN myWindow, RGB(255,0,255)
PRINT myWindow,"Test Text after frontpen used with RGB"

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
    ENDSELECT
RETURN
```

Uses RGB as a parameter for FRONTPEN to change the text color to pink.

See Also

FRONTPEN; SETCONTROLCOLOR

RIGHT$()

Usage

```
RIGHT$(string, count)
```

Parameters

string	the original string to take the characters from.
characters	the number of characters to return from the string.

Returns

A string.

Description

Returns 'count' number of character from a string starting at the end of the string.

Examples

```
OPENCONSOLE
name = "Dave Normal"
PRINT LEFT$(6, name)
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The example will display the last 6 characters of the string, in this case, "Normal".

See Also

LEFT$; MID$

RND()

Usage

```
RND(n)
```

Parameters

n	a number

Returns

A number.

Description
Returns a randomly generated number from the range `n` starting from 0. For instance, RND(9) will return a number from 0 to 8 inclusive.

Examples
```
OPENCONSOLE
PRINT RND(20)+1
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Prints out a random number from 1 to 20 inclusive.

RTRIM$()

Usage
```
RTRIM$( string )
```

Parameters
`string` a string.

Returns
A string.

Description
Removes any trailing whitespace characters from a string. Returns the resulting string leaving the original unchanged.

Examples
```
DEF Result$ AS STRING
Result$ = "Hello          "
PRINT "String Length: (trailing spaces)"
PRINT LEN(Result$)
Result$ = RTRIM$(Result$)
PRINT "New length:"
PRINT LEN(Result$)
END
```

Defines the variable Result as "Hello" with trailing spaces. Prints the length of the string, removes the trailing spaces and then prints the new length of the string.

See Also
`LTRIM$`

SEEK

Usage
```
SEEK filehandle, position

-----OR-----

currentposition = SEEK(filevariable)
```

Parameters
`filehandle` a file to access. Must be a binary file (`BFILE`)
`position` the position in the file to move to.

`currentposition` the current position in the `filehandle`.

Returns
Current file position.

Description

Allows setting the file pointer to a position for reading or writing. SEEK also allows obtaining the current file position.

Examples

```
OPENCONSOLE
DEF myfile:BFILE
DEF mynumber:INT
DEF currentposition:INT
mynumber = 3000000
IF(OPENFILE(myfile,"C:\CREATIVE BASICDATA.DAT","W") = 0)
   WRITE myfile,mynumber
   CLOSEFILE myfile
ENDIF
IF(OPENFILE(myfile,"C:\CREATIVE BASICDATA.DAT","R") = 0)
   IF(READ(myfile,mynumber) = 0)
            PRINT "This is what I read: ",mynumber
            currentposition = SEEK(myfile)
            PRINT "Current position: ",currentposition

  ENDIF
  CLOSEFILE myfile
ENDIF
PRINT "Press Any Key To Close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Creates the file CREATIVE BASICDATA.DAT with the integer 3000000 wrote to it. Reads the INT from the file then reports the pointer position inside the file with the seek command.

SELECT

Usage

```
SELECT expression
```

Parameters

expression — a variable or 1 to select the first true option.

Returns

Nothing

Description

Defines the beginning of a SELECT statement. The expression is checked against the values in the CASE statements following the SELECT statement. If they are equal, the code following the CASE statement is executed.

Examples

```
'EXAMPLE 1
OPENCONSOLE
DEF Choice$: STRING
PRINT "Press some keys, Q to quit"
Choice$ = INKEY$
DO
    DO
```

```
    Choice$ = INKEY$
    UNTIL Choice$ <> ""

    SELECT Choice$
    CASE "A"
    CASE "a"
        PRINT "You pressed A!!"
    CASE "Z"
    CASE "z"
        PRINT "Z is my favorite letter!"
    CASE "Q"
    CASE "q"
        CLOSECONSOLE
        END
    DEFAULT
        PRINT "You pressed the letter ",Choice$
    ENDSELECT
UNTIL Choice$ ="q" | Choice$ ="Q"



'EXAMPLE 2
OPENCONSOLE
DEF Choice: INT
PRINT
DO
INPUT "Enter a Number, 0 to end: ", Choice
IF Choice = 0
    CLOSECONSOLE
    END
ENDIF
SELECT 1
    CASE (Choice > 10)
        PRINT "number is greater than 10"
    CASE (Choice < 10)
        PRINT "Number is less than 10"
    CASE (Choice = 10)
        PRINT "Number is equal to 10"
    ENDSELECT
PRINT
UNTIL Choice = "0"
```

In the first example we test the choice against multiple conditions using only one SELECT statement. Your conditions can be as complex as needed, only the first true condition will be executed so make sure the conditions are unique

The SELECT statement can also be used to test multiple conditions and execute the first true case. To do this use SELECT 1 as the opening statement and code each CASE statement as a condition (must be in brackets). The second example above shows this

See Also
CASE; DEFAULT

SENDMESSAGE()

Usage
```
SENDMESSAGE( winhandle, msg, wparam, lparam {,controlID} )
```

Parameters
winhandle	user defined WINDOW or DIALOG variable.
msg	numberic value.
wparam	numeric value.
lparam	can be a numeric value, string, pointer or memory variable depending on the message being sent.
controlID	optional, control ID to send message to.

Returns
Can return a value if used as a function.

Description
Sends a message to a window, dialog or control. Return dependent on message.

Examples
```
SENDMESSAGE d1,0x18d,0,"c:\",10
```
or
```
result = SENDMESSAGE (d1,0x18d,0,"c:\",10)
```

sends a message to a list box with an ID of 10.

SETCAPTION

Usage
```
SETCAPTION winhandle, text
```

Parameters
winhandle	a user defined WINDOW or DIALOG variable.
text	a string literal, which will be displayed in the title bar of winhandle.

Returns
Nothing.

Description
Changes the caption of the window or dialog to the string or string literal specified by text.

Examples
```
DEF myWindow AS WINDOW
DEF run AS INT


WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window Example", main
SETCAPTION myWindow,"hello there!"
run = 1
WAITUNTIL run = 0



CLOSEWINDOW myWindow
```

```
END
SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

Creates a window with the caption "hello there!".

SETCONTROLCOLOR

Usage

```
SETCONTROLCOLOR winhandle, controlID, foreground, background
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
controlID	the control ID to change colour of.
foreground	The new foreground colour, used by text in the control.
background	the new background colour.

Returns

Nothing.

Description

Sets the foreground (text) and background color of a control in a window or dialog. When used with a control in a dialog SETCONTROLCOLOR should be first used in response to the @IDINITDIALOG message.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window Example", main
SETCAPTION myWindow,"Help Example"
CONTROL myWindow,"B,Button,114,72,70,20,0x50000000,1"
SETCONTROLCOLOR myWindow,1,RGB(255,255,255),RGB(0,0,0)
run = 1
WAITUNTIL run = 0

CLOSEWINDOW myWindow
END
SUB main
    SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

Creates a button and sets it foreground colour to white and the background colour to black.

See Also

SETCONTROLTEXT

SETCONTROLTEXT

Usage

```
SETCONTROLTEXT winhandle, controlID, text
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
controlID	the control ID to reference.
text	the text to change / path to a bitmap file (for buttons with images on them).

Returns

Nothing.

Description

SETCONTROLTEXT is used to change the text of a control. If the control is a bitmap button or bitmap static control then the text is the path to a bitmap file.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT


WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "SetControlText
Example", main
CONTROL myWindow,"B,Button,114,72,70,20,0x50000000,1"
run = 1
WAITUNTIL run = 0


END


SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
IF (@CLASS = @IDCONTROL)
  SETCONTROLTEXT myWindow,1,"Clicked!"
ENDIF
RETURN
```

Changes the text on the button once the button is clicked.

See Also

SETCONTROLCOLOR

SETCURSOR

Usage

```
SETCURSOR winhandle, type {,imagehandle}
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
type	cursor type to change to.
imagehandle	optional. Custom cursor to load if type is @CSCUSTOM.

Returns

Nothing

Description

Changes the cursor of the window. Type can be @CSWAIT for a wait cursor, @CSARROW for the standard arrow cursor, or @CSCUSTOM for a custom cursor. Handle is valid only for @CSCUSTOM. Cursors can be loaded with the LOADIMAGE function.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window Example", main
SETCURSOR myWindow,@CSWAIT
run = 1
WAITUNTIL run = 0

END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
RETURN
```

Sets the cursor for the window to busy.

See Also

LOADIMAGE

SETFOCUS

Usage

```
SETFOCUS winhandle {,controlID}
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
controlID	optional control ID to set focus to.

Returns

Nothing

Description

Activates the window or control and sets the input focus. If controlID is specified then the control reference by controlID will receive the input focus.

Examples

```
DEF myWindow,myWindow2 AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window 1 Example",
main
WINDOW myWindow2, 50, 50, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window 2 Example",
main
PRINT myWindow,"Left mouse button to focus on me!"
PRINT myWindow2,"Right mouse button to focus on me!"
run = 1
WAITUNTIL run = 0
```

```
END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  CLOSEWINDOW myWindow2
  run = 0
ENDIF
IF @CLASS = @IDLBUTTONUP
    SETFOCUS myWindow
ENDIF
IF @CLASS = @IDRBUTTONUP
    SETFOCUS myWindow2
ENDIF
RETURN
```

Creates 2 windows, the focus toggles between them as the user presses the left and right mouse buttons.

SETFONT

Usage

```
SETFONT winhandle, font, height, weight {,flags} {,controlID}
```

Parameters

`winhandle`	a user defined `WINDOW` or `DIALOG` variable.
`font`	the font to use.
`height`	font size.
`weight`	font typeface, 400 for normal, 700 for bold.
`flags`	optional `@SFITALIC` for an italicized font or `@SFUNDERLINE` for an underlined font.
`controlID`	if specified the control ID font is changed rather than the window's font.

Returns

Nothing.

Description

Changes the font in the window or control to the one specified in typeface. A weight of 400 is a normal typeface. Use 700 and above for a bold typeface. Flags can be `@SFITALIC` for an italicized font or `@SFUNDERLINE` for an underlined font. If the optional `controlID` is specified the controls font is changed.

Examples

```
DEF myWindow : WINDOW
WINDOW myWindow, 100, 100, 300, 100, @SIZE|@MINBOX|@MAXBOX, 0, "SETFONT Demo", main
SETFONT myWindow, "Arial", 20, 700, @SFITALIC
PRINT myWindow, "ARIAL bold italic"
run = 1

WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
```

```
SELECT @CLASS
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

Sets the font of myWindow to Arial, size 20, bold and italic.

See Also

FONTREQUEST

SETHORIZEXTENT

Usage

```
SETHORIZEXTENT window, controlID, width
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
controlID	the control ID of the listbox to modify.
width	horizontal scroll width in pixels.

Returns

Nothing.

Description

Sets the amount a listbox or the listbox portion of a combo box can be scrolled horizontally. Control must have been created with the @HSCROLL style.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window 1 Example",
main
CONTROL myWindow,"L,ListBox1,81,41,115,76,@HSCROLL,1"
SETHORIZEXTENT myWindow, 1, 400
run = 1
WAITUNTIL run = 0

END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
RETURN
```

Creates a listbox control with the @HSCROLL flag, and then sets the horizontal width of the scroll bar to 400 pixels.

See Also

SETLBCOLWIDTH

SETICON

Usage

```
SETICON winhandle, icon
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
icon	the full path of the icon you want to use.

Returns
Nothing.
Description
Changes the ICON of a window. icon is an icon loaded by the LOADIMAGE function.
Examples

```
DEF myWindow AS WINDOW
DEF run AS INT
DEF icon AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window 1 Example",
main
icon = LOADIMAGE("C:\myicon.ico",@IMGICON)
SETICON myWindow,icon
run = 1
WAITUNTIL run = 0

END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
RETURN
```

Loads an icon from C:\myicon.ico change this to point to any ICO file you like. Sets the icon as the window icon.

SETID

Usage

```
 SETID identifier, value
```

Parameters

identifier	user constant idenfifier.
value	value to set the constant to.

Returns
Nothing.
Description
Creates a user constant. The constant can then be referred to with the @ symbol just like system constants. The identifier must be unique and cannot be the same as any other system constant or variable.
Examples

```
OPENCONSOLE
SETID "ISDRAWING",10
PRINT "@ISDRAWING = ",@ISDRAWING
PRINT
PRINT "Press any key to close this program"
```

```
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Creates a user defined constant called ISDRAWING, and assigns 10 to it. It can be referenced like a system constant with the @ symbol.

SETLBCOLWIDTH

Usage

```
SETLBCOLWIDTH winhandle, controlID, width
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
controlID	the ID of the multi-column list box.
width	the column widths.

Returns

Nothing.

Description

Sets the width in pixels of columns in a multi-column list box. A multi-column list box has fixed column widths.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window 1 Example",
main
CONTROL myWindow,"L,ListBox1,81,41,115,76,@CTLISTCOLUMNS,1"
SETLBCOLWIDTH myWindow,1,40
run = 1
WAITUNTIL run = 0

END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
RETURN
```

Creates a multicolumn list box and assigns the column width to 40 pixels.

See Also

SETHORIZEXTENT

SETLINESTYLE

Usage

```
SETLINESTYLE winhandle, style, width
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
style	style of line.

Name	Purpose / Style
@LSSOLID	Draw a solid line.
@LSDASH	Draw a dashed line.
@LSDASHDOT	Draw an alternating dashed dotted line.
@LSDASHDOTDOT	Draw an alternating dashed dot dot line.
@LSDOT	Draw a dotted line.
@LSINSIDE	Draws lines around objects on the inside instead of out.

width the solid line width.

Returns

Nothing.

Description

Sets the current line drawing style in the window. Style can be one of: @LSSOLID, @LSDASH, @LSDOT, @LSDASHDOT, @LSDASHDOTDOT or @LSINSIDE. Width determines the line width for solid lines.

Examples

```
DEF w:WINDOW
WINDOW w,0,0,640,220,@SIZE|@MINBOX|@MAXBOX,0,"Lines",main
CENTERWINDOW w
SETLINESTYLE w,@LSDASH,1
LINE w,4,20,620,20,RGB(255,0,0)
LINE w,620,180,RGB(0,0,255)
LINE w,4,180,RGB(0,255,0)
LINE w,4,20,RGB(255,255,0)
run = 1

WAITUNTIL run = 0
CLOSEWINDOW w
END
SUB main
SELECT @CLASS
     CASE @IDCLOSEWINDOW
                run = 0
ENDSELECT
RETURN
```

Draws 4 dashed lines with width 1.

See Also

LINE

SETPRECISION

Usage

```
 SETPRECISION value
```

Parameters

value the number of decimal places to be displayed by PRINT.

Returns

Nothing.

Description

Sets the number of decimal places displayed by the PRINT statement for FLOAT and DOUBLE variable types. Defaults at 2.

Examples

```
OPENCONSOLE
SETPRECISION 3
PRINT 100.018746 + 3450.345 + 13.2174
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

PRINT command is set to 3 decimal place precision.

See Also

PRINT

SETSCROLLPOS

Usage

```
SETSCROLLPOS winhandle, controlID, position
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
controlID	the control ID of the scrollbar.
position	the position to set the slider at.

Returns

Nothing.

Description

Sets the slider position of a scrollbar. If ID = -1 then sets the scroll position of the windows horizontal scrollbar. If ID = -2 then sets the scroll position of the windows vertical scrollbar. Any other ID value is a user-defined scrollbar.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT


WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX | @HSCROLL | @VSCROLL, 0,
"Window 1 Example", main


SETSCROLLRANGE myWindow,-1,0,300
SETSCROLLRANGE myWindow,-2,0,300


SETSCROLLPOS myWindow,-1,50


run = 1
WAITUNTIL run = 0


END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
RETURN
```

Sets up a window with two scroll bars and moves the horizontal one along by 50.

See Also

SETSCROLLRANGE; GETSCROLLPOS

SETSCROLLRANGE

Usage

```
SETSCROLLRANGE winhandle, controlID, min, max
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
controlID	the control ID of the scrollbar.
min	the min value for the scroller.
max	the max value for the scroller.

Returns

Nothing.

Description

Sets the minimum and maximum range of a scrollbar control to min and max. All values returned by the scrollbar will be between min and max. If ID = -1 then sets the range of the windows horizontal scrollbar. If ID = -2 then sets the range of the windows vertical scrollbar.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX | @HSCROLL | @VSCROLL, 0,
"Window 1 Example", main

SETSCROLLRANGE myWindow,-1,0,300
SETSCROLLRANGE myWindow,-2,0,300

SETSCROLLPOS myWindow,-1,50

run = 1
WAITUNTIL run = 0

END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
RETURN
```

Creates a window called myWindow with horizontal and vertical scrollbars with the range 0-300.

See Also

SETSCROLLPOS

SETSELECTED

Usage

```
SETSELECTED winhandle, controlID, item
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
controlID	control ID of a list or combo box.
item	the item to select.

Returns

Nothing.

Description

Sets the currently selected item in a list or combo box control.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX | @HSCROLL | @VSCROLL, 0,
"SetSelected Example", main

CONTROL myWindow,"L,ListBox1,115,29,72,121,0x50800140,1"

ADDSTRING myWindow,1,"0"
ADDSTRING myWindow,1,"1"
ADDSTRING myWindow,1,"2"

SETSELECTED myWindow,1,1

run = 1
WAITUNTIL run = 0

END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
RETURN
```

Displays a list box with 3 items and selects the 2nd one as default.

See Also

ISSELECTED; GETSELECTED

SETSIZE

Usage

```
SETSIZE winhandle, left, top, width, height {,controlID}
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
left	position of the window.
top	position of the window.
width	window width.
height	window height.
controlID	optional, control to reference.

Returns

Nothing.

Description
Changes the size of a window, dialog, or control. If ID is specified then it must be a valid and visible control.

Examples
```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window Example", main
SETCAPTION myWindow,"SetSize Example"
CONTROL myWindow,"B,Button,114,72,70,20,0x50000000,1"
run = 1
WAITUNTIL run = 0

END

SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
IF (@CLASS = @IDCONTROL)
  SETSIZE myWindow,0,0,600,460
ENDIF
RETURN
```

Resizes the window when the button is clicked.

See Also
GETSIZE

SETSTATE

Usage
```
SETSTATE winhandle, controlID, state
```

Parameters
winhandle	a user defined WINDOW or DIALOG variable.
controlID	control ID of radio button or checkbox.
state	state of the control, 1 = checked, 0 = unchecked.

Returns
Nothing.

Description
Check/uncheck a radio button or checkbox control. State = 1 for checked, 0 for unchecked.

Examples
```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window Example", main
SETCAPTION myWindow,"SetState Example"
CONTROL myWindow,"B,On,75,72,70,20,0x50000000,1"
CONTROL myWindow,"B,Off,150,72,70,20,0x50000000,2"
```

```
CONTROL myWindow,"C,Check,103,112,70,20,0x50000003,3"

run = 1
WAITUNTIL run = 0

END

SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
IF (@CLASS = @IDCONTROL) & (@CONTROLID = 1)
  SETSTATE myWindow,3,1
ENDIF
IF (@CLASS = @IDCONTROL) & (@CONTROLID = 2)
  SETSTATE myWindow,3,0
ENDIF
RETURN
```

Creates a window with a check box and two buttons to modify the state, one turns state of the checkbox to ticked and one to unticked.

See Also

GETSTATE

SETUSERDATA

Usage

```
SETUSERDATA winhandle, variable
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
variable	data to be stored.

Returns

Nothing.

Description

Stores temporary data in the window. Variable can be INT, UINT or POINTER type.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT
DEF x AS INT
x = 9
WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window Example", main
SETUSERDATA myWindow,x
GETUSERDATA myWindow,x
PRINT myWindow,x

run = 1
WAITUNTIL run = 0
```

```
END

SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
RETURN
```

Defines an integer 'x', saves the data in the window then retrieves it and prints it to the window.

See Also

GETUSERDATA

SETUSERDATA

Usage

```
SETUSERDATA window, variable
```

Parameters

window the window to reference

variable data to be stored

Returns

nothing

Description

Stores temporary data in the window. Variable can be INT, UINT or POINTER type.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT
DEF x AS INT
x = 9
WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window Example", main
SETUSERDATA myWindow,x
GETUSERDATA myWindow,x
PRINT myWindow,x

run = 1
WAITUNTIL run = 0

END

SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
RETURN
```

Defines an integer 'x', saves the data in the window then retrieves it and prints it to the window.

See Also

GETUSERDATA

SETWINDOWCOLOR

Usage

```
SETWINDOWCOLOR winhandle, color
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
color	the background colour. The RGB function can be used to set this value.

Returns

Nothing.

Description

Changes the background color of the window.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window 1 Example",
main
SETWINDOWCOLOR myWindow,RGB(0,0,255)
run = 1
WAITUNTIL run = 0

END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
RETURN
```

Creates a window and sets its background colour to blue: RGB(0,0,255).

See Also

RGB

SGN()

Usage

```
SGN(n)
```

Parameters

n	a number.

Returns

A number.

Description

Returns -1 for a negative number 1 for a positive number and 0 for zero.

Examples

```
OPENCONSOLE
PRINT "SGN(4) = ",SGN(4)
PRINT "SGN(4) = ",SGN(0)
```

```
PRINT "SGN(4) = ",SGN(-4)
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Defines 3 variables, x y and z and applies `SGN` to them all to determine if they are positive, negative or zero.

SHOWDIALOG

Usage

```
 SHOWDIALOG dialog
```

Parameters

`dialog` the dialog to display.

Returns

Nothing.

Description

Displays a non modal dialog.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT
DEF d1 as DIALOG

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Dialog Example", main
DIALOG d1,0,0,295,174,0x80C80080,0,"Dialog!",main
CONTROL myWindow,"B,CLICK,114,72,70,20,0x50000000,1"



run = 1
WAITUNTIL run = 0

END
SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEDIALOG d1
  CLOSEWINDOW myWindow
  run = 0
ENDIF
IF (@CLASS = @IDCONTROL)
  SHOWDIALOG d1
ENDIF
RETURN
```

Shows a dialog once the button is clicked, the dialog uses the same handler as the window.

See Also
CLOSEDIALOG

SHOWIMAGE

Usage
```
SHOWIMAGE winhandle, image, type, x, y{,imagewidth ,imageheight {,flags}}
```

Parameters
winhandle	a user defined WINDOW variable.
image	the image to show.
type	the same value specified in the LOADIMAGE statement.
x	upper left corner of the image.
y	upper left corner of the image.
imagewidth	the width of the image.
imageheight	the height of the image.
flags	optional. For advanced users, for passing directly to the Windows function BitBlt.

Returns
Nothing.

Description
Draws a previously loaded IMAGE, ICON or EMF in the window. X,Y specify the upper right corner of the image. W, H specify the width and height. The optional flags variable is for advanced use and are passed directly to the WIN32 BitBlt function. The flags variable is only valid for bitmaps. Types are the same as indicated for the LOADIMAGE function. W and H are Ignored for Icons.

Examples
```
DEF myWindow AS WINDOW
DEF run AS INT
DEF pic AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window Example", main
SETCAPTION myWindow,"Show Image Example"
pic = LOADIMAGE("C:\mypic.bmp",@IMGBITMAP)
CONTROL myWindow,"B,Button,114,72,70,20,0x50000000,1"
run = 1
WAITUNTIL run = 0

END

SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run = 0
ENDIF
IF (@CLASS = @IDCONTROL)
  SHOWIMAGE myWindow,pic,@IMGBITMAP,0,0,100,100
ENDIF
RETURN
```

Loads an image from C:\mypic.bmp (please place a bmp file here) once the button is pressed.

See Also
LOADIMAGE

SHOWWINDOW

Usage

```
SHOWWINDOW winhandle, state {,controlID}
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
state	the window state to use.
controlID	optional. The ID number of a control to show.

Returns

Nothing.

Description

Sets the windows or controls state to one of the following:
@SWRESTORE, @SWMINIMIZED, @SWMAXIMIZED, @SWHIDE

Examples

```
DEF myWindow,myWindow2 AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window Example", main
WINDOW myWindow2, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Window Example", main

PRINT myWindow2,"Minimize this window"

CONTROL myWindow,"B,CLICK ME,104,48,70,20,0x50000000,2"



run = 1
WAITUNTIL run = 0

END

SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  CLOSEWINDOW myWindow2
  run = 0
ENDIF
IF (@CLASS = @IDCONTROL)
  SHOWWINDOW myWindow2,@SWRESTORE
ENDIF
RETURN
```

Creates two windows, the first has a button which when pressed uses SHOWWINDOW to restore the second window to the screen after being minimized.

See Also

CLOSEWINDOW

SIN()

Usage

```
SIN(n)
```

Parameters

n a number.

Returns

A number.

Description

Returns the SINe of the numeric expression 'n'.

Examples

```
OPENCONSOLE
PRINT "SIN(50) =", SIN(50)
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Prints the sine of 50.

See Also

SINH

SINH()

Usage

```
SINH(n)
```

Parameters

n a number.

Returns

A number.

Description

Returns the Hyperbolic SINe of the numeric expression 'n'.

Examples

```
OPENCONSOLE
PRINT "SINH(50) = ", SINH(50)
PRINT
PRINT "Press any key to end this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Prints the hyperbolic sine of 50.

See Also

TAN; COS; SIN; TANH; COSH

SPACE()

Usage

```
SPACE$(n)
```

Parameters

n a number.

Returns

A string.

Description

Returns a string filled with n spaces.

Examples

```
OPENCONSOLE
DEF myString AS STRING
myString = SPACE$(5)
PRINT "This space was",myString,"created using the SPACE$() command"
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Creates a variable named myString and assigns 5 spaces to it.

See Also

STRING

SQRT()

Usage

```
SQRT(n)
```

Parameters

n	a number.

Returns

A number.

Description

Returns the SQuare RooT of the numeric expression 'n'.

Examples

```
OPENCONSOLE
PRINT "SQRT(25) = ",SQRT(25)
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Prints the square root of 25.

STARTTIMER

Usage

```
STARTTIMER winhandle, count {,timerID}
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
count	how often to send @IDTIMER.
timerID	optional timer ID, for multiple timers.

Returns

Nothing.

Description

Starts the windows timer. Window will send a timer event every 'time' milliseconds. This event will be sent to the windows main procedure subroutine. The optional ID lets you specify multiple timer events in a window. The default ID is 1. Windows will send an `@IDTIMER` message to your handler subroutine with the id of the timer in `@CODE`.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Timer Example", main
STARTTIMER myWindow,1000
CONTROL myWindow,"B,Button,114,72,70,20,0x50000000,1"
run=1
WAITUNTIL run=0

END

SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run=0
ENDIF
IF @CLASS = @IDTIMER
  PRINT myWindow,"."
ENDIF
ENDIF
IF (@CLASS = @IDCONTROL)
  STOPTIMER myWindow
ENDIF
RETURN
```

Prints a row of .'s every 1000 millisecs by checking for `@IDTIMER` inside the handler.

See Also

STOPTIMER

STEP

Usage

```
 FOR loopvar = startno TO endno #stepno

----- OR -----


FOR loopvar = startno TO endno STEP stepno
```

Parameters

`loopvar`	a variable to hold the counter.
`startno`	the starting number.
`endno`	the ending number.
`stepno`	optional. The amount to increment each loop, can be negative (decrement).

Returns
Nothing. The variable `loopvar` will be changed by the `FOR` loop.

Description
Step, a positive number, can be used to set an increment value (or negative for a decrement by value). If step is left out, the loop increments by 1 each loop. This type of loop is best suited for situtions when you know which values you want to use, for example setting the ID's of 10 controls. It would not be suitable for finding the names of every file in a directory, since you don't know how many files there may be.

Examples
```
REM ----- Example 1 -----
OPENCONSOLE
DEF loopcounter: INT
PRINT "Counting the numbers from 1 to 10"
FOR loopcounter = 1 to 10
    PRINT loopcounter
NEXT loopcounter
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END

REM ----- Example 2 -----
OPENCONSOLE
DEF loopcounter: INT
PRINT "Counting the even numbers from 10 down to 1"
FOR loopcounter = 10 to 1 STEP -2
    PRINT loopcounter
NEXT loopcounter
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Example 1 simply prints out the numbers 1 to 10. Note how there is no `STEP` part to the loop. When this isn't present, Creative BASIC automatically increments the `loopvar` by 1.

Example 2 uses the `STEP` part to print out all the even numbers between 1 and 10, starting at 10 and working backward through 8,6,4,2 etc.

See Also
NEXT; FOR

STOPTIMER
Usage
```
STOPTIMER winhandle {,controlID}
```

Parameters

winhandle	a user defined WINDOW or DIALOG variable.
controlID	optional, the id number of a control.

Returns

Nothing.

Description

Stops a previously started timer in a window.

Examples

```
DEF myWindow AS WINDOW
DEF run AS INT

WINDOW myWindow, 0, 0, 320, 240, @SIZE | @MINBOX | @MAXBOX, 0, "Timer Example", main
STARTTIMER myWindow,1000
CONTROL myWindow,"B,Button,114,72,70,20,0x50000000,1"
run=1
WAITUNTIL run=0

END

SUB main
IF @CLASS = @IDCLOSEWINDOW
  REM closes the window and sets w1 = 0
  CLOSEWINDOW myWindow
  run=0
ENDIF
IF @CLASS = @IDTIMER
  PRINT myWindow,"."
ENDIF
ENDIF
IF (@CLASS = @IDCONTROL)
  STOPTIMER myWindow
ENDIF
RETURN
```

Stops the timer once the button is pressed.

See Also

STARTTIMER

STR$()

Usage

```
STR$(n)
```

Parameters

n	a number.

Returns

A string.

Description

Returns a string representation of a number.

Examples

```
OPENCONSOLE
```

```
DEF x,y AS STRING
x = STR$(6)
y = STR$(2)
PRINT x,y

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Creates two variables called x and y and assigns two string representations of numbers to them.

See Also

VAL

STRING()

Usage

```
STRING$( number, char )
```

Parameters

number	the string length.
char	the characters to fill the string with.

Returns

A string.

Description

Returns a string of length number filled with char.

Examples

```
OPENCONSOLE
PRINT "STRING$(4,"x") = ",STRING$(4,"x")
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Defines a string variable x, and assigns 4 x's to its contents.

See Also

SPACE

SUB

Usage

```
SUB name {, parameterlist}
```

Parameters

name	the name of the subroutine.
parameterlist	user defined parameters.

Returns

User defined.

Description

Defines a subroutine. If a subroutine is declared with the DECLARE statement, an optional parameter list may be supplied. Exit from a subroutine using the RETURN statement.

Examples

```
DECLARE calculate(x: FLOAT ,y: FLOAT )
```

```
OPENCONSOLE
PRINT calculate(25,75)
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END

SUB calculate(x,y)
DEF ret: FLOAT
ret = x + y

RETURN ret
```

A subroutine called calculate which takes two FLOATs as input, adds them together and returns a FLOAT as output.

See Also

RETURN; LABEL; DECLARE; RETURN

SYSTEM

Usage

```
SYSTEM command {, parameter}
```

Parameters

command	the path to the system program to run.
parameter	optional, parameter to send to the program to be executed.

Returns

Nothing.

Description

Executes another program. Command must be the fully qualified pathname to the executable or the executable must exist in a system directory. The optional parameter is sent to the program to be executed.

Examples

```
OPENCONSOLE
SYSTEM "notepad.exe", "c:\windows\setuplog.txt"
PRINT "Notepad should now have opened a file!"
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Loads up notepad and sends to it the parameter to load c:\windows\setuplog.txt.

See Also

GETSTARTPATH;

TAN()

Usage

```
TAN(n)
```

Parameters

`n` a numeric expression

Returns

A number.

Description

Returns the TANgent of the numeric expression `n`.

Examples

```
OPENCONSOLE
PRINT "TAN(2) =", TAN(2)
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Returns -2.19

See Also

TANH; COS; SIN; COSH; SINH

TANH()

Usage

```
TANH(n)
```

Parameters

`n` a numeric expression.

Returns

A number.

Description

Returns the Hyperbolic TANgent of the numeric expression `n`.

Examples

```
OPENCONSOLE
PRINT "TANH(2) =", TANH(2)
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Returns the value 0.96

See Also

TAN; COS; SIN; COSH; SINH

TIME$

Usage

```
TIME$
```

Parameters

None.

Returns

The current System time as a string.

Description

Returns the current system time in the format HH:MM:SS.

Examples

```
OPENCONSOLE
PRINT "The current time ", TIME$
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Assigns the current time to a string and prints it to the screen.

See Also

DATE$()

TIMER

Usage

```
TIMER
```

Parameters

None.

Returns

A number.

Description

Returns the number of seconds elapsed since Windows was started.

Examples

```
OPENCONSOLE
PRINT "Windows has been running for ", TIMER, "seconds"
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Prints the windows uptime to the screen.

TO

Usage

```
FOR counter = start TO stop
```

Parameters

counter	an integer variable.
start	the counter start value.
stop	the counter stop value.

Returns

Nothing.

Description
Used with FOR statement.
Examples
```
OPENCONSOLE
DEF counter:INT
DEF name$:STRING

INPUT "Enter your name ",name$

FOR counter = 1 TO 20 #2
    PRINT "Hello ",name$
NEXT counter

PRINT "Press Any Key To Close"
DO:UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Increments the variable counter through 1 to 20, printing out some text each time.
See Also
FOR; TO

TYPE
Usage
```
 TYPE name {,pack}
```
Parameters
name	a user defined variable type.
pack	optional, how it will be stored in memory (for DLL functions).

Returns
Nothing.
Description
Begins the declaration of a user defined variable type. The variables members can be any built-in or user defined type.
Examples
```
OPENCONSOLE
'create a new Creative BASIC variable type
TYPE contactdetails
   DEF Name:STRING
   DEF Age:INT
   DEF Phone[20]:ISTRING
ENDTYPE
'declare a variable of type contactdetails
DEF contact:contactdetails

'fill in the details of your contact into the variable
contact.Name = "Joe Smith"
contact.Age = 35
contact.Phone = "555-555-1212"
```

```
'Print the contact's details out
PRINT "Contact's Name:  ",contact.Name
PRINT "Contact's Age:   ",contact.Age
PRINT "Contact's Phone: ",contact.Phone
PRINT
PRINT "Press a key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program defines a new variable type (contactdetails) and uses a variable of this type (contact) to store and print a telephone contact's details.

See Also

ENDTYPE

TYPEOF()

Usage

```
TYPEOF(pointer)
```

Parameters

`pointer` the pointer referece to check.

Returns

An integer specifying the type of the variable the pointer is referencing.

Description

Returns an integer value specifying the type of the variable the pointer is referencing.

Examples

```
OPENCONSOLE
DEF myPointer:POINTER
DEF myNumber:INT

myNumber = 55
myPointer = myNumber

PRINT #myPointer
id = TYPEOF(myPointer)
IF id = @TYPEINT
    PRINT "INTEGER POINTER"
ENDIF

DO
UNTIL INKEY$<> ""
CLOSECONSOLE
```

Assigns a pointer to the 'i' variable, and determines that it is of type integer.

UCASE$()

Usage

```
UCASE$(string)
```

Parameters
`string` a string.
Returns
A string, the uppercase version of the `string` parameter.
Description
Returns the string parameter converted to all uppercase letters.
Examples

```
OPENCONSOLE
PRINT "UCASE$("hello") = ",UCASE$("hello")
PRINT
PRINT "Press any key to close this program"

DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Converts a string into uppercase.
See Also
LCASE

UNTIL

Usage

```
 UNTIL condition
```

Parameters
`condition` any valid condition.
Returns
Nothing.
Description
Ends a `DO` loop once the condition is true, until then, the code between the `DO` and `UNTIL` statements is looped.
Examples

```
OPENCONSOLE
DEF x:INT
x=1

DO
    PRINT x
    x=x+1
UNTIL x > 10

PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Prints and then increments the variable x until x is greater than 10.
See Also
DO; WHILE; FOR

USING()

Usage

```
USING (format, userString{,userString...})
```

Parameters

`format` — a string literal or variable that can contain special formatting symbols as well as regular text to be inserted into the final output string.

Symbol	Meaning
#	Reserves a place for one digit
Point (.)	Determines decimal point locations
Minus (-)	Left justifies within field. Default is right.
0	Prints leading zeros instead of spaces. Ignored if used with left justification.
Comma (,)	Prints a comma before every third digit to the left of the decimal point and reserves a place for one digit or digit separator.
&	Copies the string parameter directly

`userString` — the string to be formatted.

Returns

A string, formatted as specified.

Description

Returns a string that is formatted based on a specifier string and one or more parameters.

Examples

```
OPENCONSOLE
PRINT USING("$#,###.##", 1145.551)
PRINT
PRINT USING("&$#,###.##&", "The total cost is ", 3000.55, " Dollars")
PRINT
A$ = USING("$#####.##", 22.8)
PRINT A$
PRINT
PRINT "Press any key to close this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

The above program simply prints out some currency values using the `USING` command.

See Also

PRINT

VAL()

Usage

```
VAL(string)
```

Parameters

`string` — a string containing a numeric value.

Returns

The content of `string` as a numeric value.

Description

Converts the value stored in a string to a numeric value.

Examples

```
DEF myString:STRING
DEF myInt:INT
OPENCONSOLE
myString = "54321"
myInt = VAL(myString)
```

```
PRINT myInt
PRINT "Press any key to end this Program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

"54321" is stored as a string (myString), and then converted to a numeric value (myInt).

See Also

STR

WAIT

Usage

```
WAIT {NoSleep}
```

Parameters

NoSleep	if set to 1 then the WAIT command will check for messages and return immediately.

Returns

Nothing

Description

Waits for a window event. You must have at least one open window or dialog before executing wait . When an event occurs the window/dialog main procedure is called as a subroutine. This function is for special cases and WAITUNTIL should be used normally. Optional parameter allows using WAIT inside loops.

Examples

```
DEF loopcount, cancel AS INT
OPENCONSOLE

FOR loopcount = 1 TO 1000
    REM Complex math and drawing code here
   WAIT 1
   IF cancel = 1 THEN loopcount = 1000
NEXT x

CLOSECONSOLE
END
```

This example assumes that there is some menu option, or button that sets the variable cancel equal to 1. If you did not use WAIT 1 in this case the program would appear to be 'locked' and would not respond to menu's or buttons until the long loop was finished. You should limit your use of WAIT 1 since this creates a "busy wait " loop. Since your program is never allowed to sleep it consumes the majority of the processor time on the system. This will reduce overall system performance.

See Also

WAITUNTIL

WAITUNTIL

Usage

```
WAITUNTIL condition
```

Parameters

condition	any valid condition.

Returns

Nothing.

Description

Processes window events until condition is true. You must have at least one open window or dialog before executing WAITUNTIL . When an event occurs the window/dialog main procedure is called as a subroutine.

Examples

```
REM define a window variable
DEF myWindow:WINDOW
REM open the window
WINDOW myWindow,0,0,350,350,@MINBOX|@MAXBOX|@SIZE,0,"Simple Window",main
REM print a message
PRINT myWindow,"Hello World"
REM when myWindow = 0 the window has been closed
WAITUNTIL myWindow = 0
END
'---
REM every time there is a message for our window
REM the operating system will GOSUB here
SUB main
    IF @CLASS = @IDCLOSEWINDOW
        REM closes the window and sets myWindow = 0
        CLOSEWINDOW myWindow
    ENDIF
RETURN
```

The example shows the steps necessary to open a window and wait for a message. If you examine the program, you will notice that there are no GOSUB statements. Windows will call your subroutine anytime there is a message for the window you created.

See Also

WAIT; GOSUB; WAITUNTIL

WHILE

Usage

```
WHILE condition
```

Parameters

condition — any valid condition.

Returns

Nothing.

Description

Begins a WHILE loop. As long as the condition is true, all of the statements between WHILE and ENDWHILE are executed in a loop. A WHILE loop checks condition at the start of every loop, therefore the program may not execute the code contained in the loop at all.

Examples

```
OPENCONSOLE
DEF x:INT
x=1

WHILE x < 11
    PRINT x
    x=x+1
```

```
ENDWHILE

PRINT "Press Any Key To Close"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Prints X, then increments X and loops, while the condition X < 11 is true.

See Also

ENDWHILE; DO; UNTIL

WINDOW

Usage

```
WINDOW winvariable, L, T, W, H, flags, parent, title, handler
```

Parameters

`winvariable` variable name of the `WINDOW` variable used to store the window.

`L` left edge of the window.

`T` top edge of the window.

`W` width of the window.

`H` height of the window.

`flags` numeric value, specifies style creation flags.

Name	Purpose / Style
@SIZE	Creates a window that is resizable
@MINBOX	The window has a minimize box
@MAXBOX	The window has a maximize box
@MINIMIZED	Creates a window that is initially minimized
@MAXIMIZED	Creates a window that is initially maximized
@CAPTION	Default. Creates a window with a caption
@NOCAPTION	Creates a window with no caption
@SYSMENU	Default. Creates a standard system menu
@BORDER	Creates a bordered window. Use with the @NOCAPTION flag
@HSCROLL	Window has a horizontal scroll bar
@VSCROLL	Window has a vertical scroll bar
@MDIFRAME	Creates a window that can contain other windows.
@USEDEFAULT	Child windows can use this for the 'left' parameter to let windows pick a default size for the window
@TOOLWINDOW	Creates a window with a half sized caption.
@NOAUTODRAW	Allows your program to handle @IDPAINT messages
@BROWSER	Creates a window with an embedded internet explorer window.

`parent` `WINDOW` variable of parent window if this window is a child, otherwise 0.

`title` the text to be shown in the caption of the window.

handler the name of a subroutine to handle messages for the window.

Returns

Nothing.

Description

Opens a window with the dimensions, flags and title specified. Handler is the name of the subroutine to catch the windows events. If parent is specified then this will be a child window of that window or dialog.

Examples

```
DEF myWindow:WINDOW
WINDOW myWindow,0,0,640,200,@SIZE|@MINBOX|@MAXBOX,0,"TEST",main

run = 1
WAITUNTIL run = 0
CLOSEWINDOW myWindow
END

SUB main
SELECT @CLASS
    CASE @IDCONTROL
    CASE @IDCLOSEWINDOW
        run = 0
ENDSELECT
RETURN
```

The above example opens a demo window.

See Also

CLOSEWINDOW

WRITE()

Usage

```
WRITE (filehandle, data {,data.})
```

Parameters

filehandle the file opened with the OPENFILE command.

data data to be written to the open file.

Returns

0 on success.

Description

Writes out data to a file. filehandle must have been opened with the OPENFILE function.

Examples

```
DEF myfile:FILE
OPENCONSOLE
IF(OPENFILE(myfile,"c:\writetest.txt","W") = 0)
   WRITE myfile,"This is a write test"
   CLOSEFILE myfile
   PRINT "File created successfully"
ELSE
   PRINT "File could not be created"
ENDIF

PRINT "Press any key to end this program"
```

```
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Creates a file and writes "This is a test write" to the file, and then closes it.

See Also

READ; OPENFILE

WRITEMEM

Usage

```
WRITEMEM memory, position, variable {,offset}
```

Parameters

memory	variable of type memory, allocated with the ALLOCMEM command.
position	must be greater than 0.
variable	any built in, or user-defined variable type.
offset	allows you to specify the position where reading and writing will occur. Amount is in bytes.

Returns

Nothing.

Description

Writes one record to memory. position must be greater than 0. variable can be any built-in or user defined variable type. memory must have been successfully allocated with the ALLOCMEM statement. Similar to the PUT statement.

Examples

```
REM example of memory functions
OPENCONSOLE
DEF num:INT
DEF buffer:MEMORY
DEF buffer2:MEMORY
TYPE mytype
                DEF name[40]:ISTRING
                DEF age:INT
                DEF money:FLOAT
ENDTYPE
DEF record:mytype
ALLOCMEM buffer,100,LEN(num)
ALLOCMEM buffer2,10,LEN(record)


FOR x = 1 to 100
WRITEMEM buffer,x,x
NEXT x

FOR x = 1 to 100
READMEM buffer,x,num
PRINT num
NEXT x
```

```
record.name = "John Doe"
record.age = 32
record.money = 100.01

WRITEMEM buffer2,1,record
READMEM buffer2,1,record
PRINT record.name
PRINT record.age
PRINT record.money
FREEMEM buffer
FREEMEM buffer2

PRINT "Press any key to end this program"
DO
UNTIL INKEY$ <> ""
CLOSECONSOLE
END
```

Creates a type, allocates memory, writes it to memory and then reads it from memory.

See Also

ALLOCMEM; FREEMEM; PUT

WRITEPRINTER

Usage

```
WRITEPRINTER printerhandle, data {,data2}
```

Parameters

printerhandle	a printer opened with the OPENPRINTER function.
data	the data to be printed. This can be a variable or a literal string.
data2	optional. The data to be printed. This can be a variable or a literal string.
data	the data to be printed. This can be a variable or a literal string.

Returns

Nothing.

Description

Outputs data to the printer. handle must have been opened with the OPENPRINTER function.

Examples

```
OPENCONSOLE
DEF hPrt:INT
DEF data:STRING
DEF name:STRING
DEF pagefrom,pageto,copies,collate:INT
pagefrom = 1
pageto = 1
copies = 1
name = PRTDIALOG(0,pagefrom,pageto,copies,collate)
hPrt = OPENPRINTER(name,"Test Document","TEXT")
IF (hPrt)
    data = "This is a test of printing"
```

```
    data = data + chr$(13)
    data = data + "This is line 2"
    WRITEPRINTER hPrt,data
    CLOSEPRINTER hPrt
ENDIF
CLOSECONSOLE
END
```

Opens up the print dialog ready to print a test page.

Note: Not all printers support printing text directly. If your printer is referred to as a 'GDI only' printer then you may have trouble using WRITEPRINTER. Consult you printer documentation for more information.

See Also

OPENPRINTER; ENDPAGE; CLOSEPRINTER; PRTDIALOG; GETDEFAULTPRINTER; PRINTWINDOW