The Gambas Language Encyclopaedia

version 0.57 of 05/25/2003

Contents

Abs

Value = **Abs (** *Number* **)**

Computes the absolute value of a number.

Example :

```
PRINT Abs(-2)
⇨ 2
PRINT Abs(0)
⇨ 0
```

Acs / ACos

value = **Acs (** *Number* **)**
value = **ACos (** *Number* **)**

Computes the arc-cosine of a number.

Example :

```
PRINT Acs(0.5)
⇨ 1.047197551197
PRINT Acs(-1)
⇨ 3.14159265359
```

Acsh / ACosh

value = **Acsh (** *Number* **)**
value = **ACosh (** *Number* **)**

Computes the hyperbolic arc-cosine of a number.

Example :

```
PRINT Acsh(2)
⇨ 1.316957896925
```

AND

Result = *Expression* **AND** *Expression*

Computes the logical *and* of two boolean expressions, or the numerical *and* of two integer numbers.

Example :

```
PRINT TRUE AND FALSE
⇨ False
PRINT TRUE AND TRUE
⇨ True
PRINT 7 AND 11
⇨ 3
```

Arithmetic operators

Number **+** *Number*	Adds two numbers.
Number **-** *Number*	Substracts two numbers.
Number ***** *Number*	Multiplies two numbers.
Number **/** *Number*	Divides two numbers.
Number **^** *Power*	Raises *Number* to the power *Power*.
Number **** *Number*	Computes the quotient of the two numbers.
Number **MOD** *Number*	Computes the remainder of the quotient of the two numbers.
- *Number*	Computes the opposite of a number.

Array

array = **Array (** *Expression* **, ...)**

Creates an array and returns it. The type of the array is the type of the first expression. The other expressions are automatically converted.

You can use the square bracket syntax as an alternative to the **Array** subroutine.

Example :

```
PRINT Object.Type(Array(2.4, 3, 3.2))
⇨ Float[]

PRINT Object.Type(Array("2.4", 3, 3.2))
⇨ String[]

PRINT [ "A", "B", "C" ].Join("/")
⇨ A/B/C
```

Asc

Code = **Asc (** *String* [**,** *Position*] **)**

Returns the ASCII code of the character at position *Position* in the string. If *Position* is not specified, the ASCII code of the first character is returned.

Example :

```
PRINT Asc("Gambas")
⇨ 71
PRINT Asc("Gambas", 3)
⇨ 109
```

Asn / ASin

value = **Asn (** *Number* **)**
value = **ASin (** *Number* **)**

Computes the arc-sine of a number.

Example :

```
PRINT Asn(0.5)
⇨ 0.523598775598
PRINT Asn(-1)
⇨ -1.570796326795
```

Asnh / ASinh

value = **Asnh (** *Number* **)**
value = **ASinh (** *Number* **)**

Computes the hyperbolic arc-sine of a number.

Example :

```
PRINT Asnh(2)
⇨ 1.443635475179
```

Assignment

Variable = *Expression*

Assigns the value of an expression to one of the following elements :

- A local variable.
- A function parameter.
- A global (class) variable.
- An array slot.
- An object public variable.
- An object property.

Atn / ATan

value = **Atn (** *Number* **)**
value = **ATan (** *Number* **)**

Computes the arc-tangent of a number.

Example :

```
PRINT Atn(0.5)
⇨ 0.463647609001
```

Atnh / ATanh

value = **Atnh** (*Number*)
value = **ATanh** (*Number*)

Computes the hyperbolic arc-tangent of a number.

Example :

```
PRINT Atnh(0.5)
⇨ 0.549306144334
```

BChg

Value = **BChg** (*Number* , *Bit*)

Returns *Number* with its *Bit*th bit inverted.

Example :

```
PRINT BChg(15, 1)
⇨ 13
PRINT BChg(13, 1)
⇨ 15
```

BClr

Value = **BClr** (*Number* , *Bit*)

Returns *Number* with its *Bit*th bit cleared.

Example :

```
PRINT BClr(15, 1)
⇨ 13
PRINT BClr(13, 1)
⇨ 13
```

Bin$

String = **Bin$** (*Number* [, *Digits*])

Gets the binary representation of a number. If *Digits* is specified, the representation is padded with unnecessary zeros so that *Digits* digits are returned.

Example :

```
PRINT Bin$(77)
⇨ 1001101
PRINT Bin$(77, 16)
⇨ 0000000001001101
```

BREAK

BREAK

Leaves a loop immediately.

BSet

Value = **BSet (** *Number* **,** *Bit* **)**

Return if the *Bit*th bit of *Number* is set.

Example :

```
PRINT BSet(13, 1)
⇨ 15
PRINT BSet(15, 1)
⇨ 15
```

BTst

Boolean = **BTst (** *Number* **,** *Bit* **)**

Returns if the *Bit*th bit of *Number* is set.

Example :

```
PRINT BTst(15, 1)
⇨ True
PRINT BTst(13, 1)
⇨ False
```

CASE

See *SELECT*.

CATCH

CATCH

This instruction indicates the beginning of the error management part of a function or procedure.

The *catch* part is executed when an error is raised between the beginning of the function execution and its end. This error can be raised by the function itself, or by any other function called during its execution, provided that this deeper function has no *catch* part itself : the deeper the *catch* part is, the more priority it has.

If an error is raised during the execution of the *catch* part, it is normally propagated. The *catch* part does not

protect itself !

If there is a *finally* part in the function, it must precede the *catch* part. See *FINALLY* for more details.

Example :

```
' Print a file to the screen

SUB PrintFile(FileName AS STRING)

  DIM hFile AS File
  DIM sLig AS STRING

  OPEN FileName FOR READ AS #hFile

  WHILE NOT EOF(hFile)

    LINE INPUT #hFile, sLig
    PRINT sLig

  WEND

FINALLY ' Always executed, even if a error raised

  CLOSE #hFile

CATCH ' Executed only if there is an error

  PRINT "Cannot print file "; FileName

END
```

CBool

Boolean = **CBool** (*Expression*)

Converts an expression into a boolean.

An expression is false if it is :

- A false boolean.
- A zero number.
- A zero length string.
- A null object.

An expression is true in all other cases.

Example :

```
PRINT CBool(0); " "; CBool(1)
⇨ False True
PRINT CBool("Gambas"); " "; CBool("")
⇨ True False
PRINT CBool(NULL)
⇨ False
```

CByte

Byte = **CByte** (*Expression*)

Converts an expression into a byte. *Expression* is first converted into an integer. Then, if this integer overflows the byte range, it is truncated.

Example :

```
PRINT CByte("17")
⇨ 17
PRINT CByte(100000)
⇨ 160
PRINT CByte(TRUE)
⇨ 255
```

CDate

Date = **CDate** (*Expression*)

Converts an expression into a date/time value. Be careful, the current localization is NOT used by this function.

Example :

```
PRINT CDate("09/06/1972 01:45:12")
⇨ 09/06/1972 01:45:12
PRINT CDate(2484515)
⇨ 05/16/2002
```

CFloat

Float = **CFloat** (*Expression*)

Converts an expression into a floating point number. Be careful, the current localization is NOT used by this function.

Example :

```
PRINT CFloat("+3.1416")
⇨ 3.1416
PRINT CFloat("1.0E+3")
⇨ 1000
```

Choose

Value = **Choose (** *Choice* **,** *Result #1* **,** *Result #2* [**,** ... **,** *Result #n*] **)**

This function returns the value of one of its *Result* arguments, according to the value of *Choice.*

If *Choice* is 1, then *Result #1* is returned. If *Choice* is 2, Result #2 is returned, and so on...

If *Choice* is lower or equal to zero, or if no *Result* value is specified for the value of *Choice*, then NULL is

returned.

Example :

```
X = 3

PRINT Choose(X, "one", "two", "three", "four")
⇨ three

PRINT IsNull(Case(X * 2, "one", "two", "three", "four"))
⇨ True
```

Chr$

Character = **Chr$** (*Code*)

Returns the character whose ASCII code is *Code*.

Example :

```
PRINT Chr$(65)
⇨ A
```

CInt / CInteger

Integer = **CInt** (*Expression*)
Integer = **CInteger** (*Expression*)

Converts an expression into an integer.

Example :

```
PRINT CInt("17")
⇨ 17
PRINT CInt(100000)
⇨ 100000
PRINT CInt(TRUE)
⇨ -1
PRINT CInt(3.6)
⇨ 3
PRINT CInt(Now)
⇨ 2484515
```

CLOSE

CLOSE #*File*

Closes an opened file.

Comparison operators

Number = *Number*	Are the two numbers equal ?
Number <> *Number*	Are the two numbers different ?
Number < *Number*	Is the first number lower than the second ?
Number > *Number*	Is the first number greater than the second ?
Number <= *Number*	Is the first number lower or equal than the second ?
Number >= *Number*	Is the first number greater or equal than the second ?

Constants

The true value.	`TRUE`
The false value.	`FALSE`
Integer numbers.	`0, 123, -32769`
Hexadecimal short signed integers.	`&H1F5, &HFFFF, &FFFF`
Hexadecimal signed integers.	`&H10BF332E, &10BF332E`
Hexadecimal unsigned integers.	`&H8000&, &HFFFF&`
Binary integers.	`&X1010010101, %101001011`
Floating point numbers.	`1.0, -5.345E+45`
String constants.	`"Hello World !"`
String constants to be translated.	`("This software is cool")`
Null constant / void string.	`NULL`

Constants declaration

(**PUBLIC** | **PRIVATE**) **CONST** *Identifier* **AS** *Datatype* = *Constant value*

This declares a class global constant.

- This constant is accessible everywhere in the class it is declared.
- If the PUBLIC keyword is specified, it is also accessible to the other classes having a reference to an object of this class.
- Constant must be booleans, integers, floating point numbers or strings.

Example :

```
PUBLIC CONST MAX_FILE AS Integer = 30
PRIVATE CONST MAGIC_HEADER AS String = "# Gambas form file"
```

CONTINUE

CONTINUE

Jumps to the next occurrence of a loop.

Example :

```
FOR I = 1 TO 10

  IF I = 1 THEN
    PRINT " One";
    CONTINUE
  ENDIF

  IF I = 2 THEN
    PRINT " Two";
    CONTINUE
  ENDIF

  PRINT I;

NEXT

PRINT

⇨  One Two 3 4 5 6 7 8 9 10
```

Conv$

Converted string = **Conv$ (** *String* **AS String ,** *Source charset* **AS String ,** *Destination charset* **AS String)**

Converts a string from one charset to another charset. A charset is represented by a string like "ASCII", "ISO-8859-1", or "UTF-8". All **GAMBAS** strings are stored in *UTF-8* format.

The conversion relies on the **iconv()** *GNU* system function.

Example :

```
DIM sStr AS String
DIM iInd AS Integer

sStr = Conv$("Gambas", "ASCII", "EBCDIC-US")
FOR iInd = 1 TO Len(sStr)
  PRINT Hex$(Asc(Mid$(sStr, iInd, 1)), 2); " ";
NEXT

⇨ C7 81 94 82 81 A2
```

Cos

value = **Cos** (*angle*)

Computes the cosine of an angle. The angle is specified in radians.

Example :

```
PRINT Cos(Pi)
⇨ -1
```

Cosh

Value = **Cosh** (*Number*)

Compute the hyperbolic cosine of a number.

Example :

```
PRINT Cosh(1)
⇨ 1.543080634815
```

CShort

Short = **CShort** (*Expression*)

Converts an expression into a short integer. *Expression* is first converted into an integer. Then, if this integer overflows the short range, it is truncated.

Example :

```
PRINT CShort("17")
⇨ 17
PRINT CShort(100000)
⇨ -31072
PRINT CShort(TRUE)
⇨ -1
```

CStr / CString

String = **CStr** (*Expression*)
String = **CString** (*Expression*)

Converts an expression into a string. Be careful, the current localization is NOT used by this function.

Example :

```
PRINT CStr(-1972)
⇨ -1972
PRINT CStr(Now)
⇨ 05/16/2002 15:08:31
PRINT CStr(Pi)
⇨ 3.14159265359
```

Datatypes

	Description	*Memory size*	*Default value*
Boolean	`True` or `False`	1 byte	`False`
Byte	`0` ... `255`	1 byte	`0`
Short	`-32768` ... `+32767`	2 bytes	`0`
Integer	`-2147483648` ... `+2147483647`	4 bytes	`0`
Float	Like the *double* datatype in *C*	8 bytes	`0.0`
Date	Date and time, each stored in an *integer*.	8 bytes	`Null`
String	A reference to a variable length string.	4 bytes	`Null`
Variant	Any datatype.	12 bytes	`Null`
Object	A anonymous reference to an object.	4 bytes	`Null`

See also *Predefined classes*.

Date

Result = **Date (** *Date/Time* **)**

Returns a date without its time component.

Example :

```
PRINT Now; " -> "; Date(Now)
⇨ 05/16/2002 15:10:59 -> 05/16/2002
```

Date = **Date (** *Year* **,** *Month* **,** *Day* [**,** *Hours* **,** *Minutes* **,** *Seconds*] **)**

Makes a date from its components.

Example :

```
PRINT Date(1972, 09, 06, 1, 5)
⇨ 09/06/1972 01:05:00
```

Day

Result = **Day (** *Date/Time* **)**

Returns the day component of a date/time value.

Example :

```
PRINT Day(Now)
⇨ 16
```

DEC

DEC *Variable*

Decrements a variable. *Variable* can be any target of an assignment, but must be numeric.

Example :

```
' First example

X = 7

DEC X

PRINT X
⇨ 6

' Second example

DIM A[3, 3] AS FLOAT

X = 2
Y = 1

A[X, Y] = PI

DEC A[X, Y]

PRINT A[X, Y]
⇨ 2.14159265359
```

DEFAULT

See *SELECT*.

Deg

Value = **Deg (** *Angle* **)**

Converts radians to degrees.

Example :

```
PRINT Deg(Pi / 2)
⇨ 90
```

DIM

See *Local variable declaration*.

Dir$

File name = **Dir$ (** *Directory* [**,** *File pattern*] **)**

Returns the name of the first file in *Directory* that matches the pattern. If no pattern is specified, any file name is returned. The pattern can contain the same generic characters than the **LIKE** operator.

File name = **Dir$ ( )**

Returns the next entry of the directory given with the first syntax of **Dir$().**

Warning ! This function is not reentrant. You cannot use it recursively.

Example :

```
' Print a directory

SUB PrintDirectory(Directory AS String)

  DIM File AS String

  File = Dir$(Directory, "*.*")

  WHILE File

    PRINT File
    File = Dir$()

  WEND

END
```

DO

DO [**WHILE** *Expression*]

Begins an infinite loop structure delimited by DO ... LOOP instructions. If WHILE is specified, the loop is stopped when *Expression* is false.

ELSE

See *IF*.

END

Indicates the end of a procedure or a function. See *Method declaration*.

END SELECT

See *SELECT*.

END WITH

See *WITH*.

Eof

Boolean = **Eof (** *File* **)**

Returns if we are at the end of the stream.

See *LINE INPUT* for an example.

Error

Result = **Error ( )**

Returns the code of the last raised error. If there was no error, 0 is returned.

Example :

```
TRY KILL FileName
IF Error THEN PRINT "Cannot remove file. Error ="; Error
```

Eval

Variant = **Eval (** *Expression* [**,** *Context*] **)**

Evaluates an expression and returns its value. This expression can use almost all operators and subroutines of **GAMBAS**.

The optional *context* is a collection that must contain the value of each undefined symbol of *Expression*.

Example :

```
DIM Context AS New Collection

Context["X"] = 2
Context["Y"] = "Gambas"

PRINT Eval("X * Len(Y)")
⇨ 12
```

EXEC

EXEC *Command* [**WAIT**] [**FOR** (**READ** | **WRITE** | **READ WRITE**) [**AS** *Variable*]

Executes a command. An internal *Process* object is created to manage the command.

The command must be specified as an array of strings containing at least one element. The first element of this array is the name of the command, and the others are optional parameters.

If WAIT is specified, then the interpreter waits for the command ending. Otherwise, the command is executed in background.

If FOR is specified, then the command input-outputs are redirected so that your program intercepts them :

- If WRITE is specified, you can send data to the command standard input via a method of the class Process. Note that you need a reference to the Process object for that.
- If READ is specified, then events will be generated each time the command sends data to its standard output streams : the event *Write* is raised when data are sent to the standard output stream, and the event *Error* is raised when data are sent to the standard error stream.

Lastly, you can get a reference to the internal Process object into a variable *Variable* by specified the keyword AS.

Example :

```
' Get the content of a directory

EXEC [ "ls", "-la" ] WAIT

' Same thing, but in background

DIM Content AS String

EXEC [ "ls", "-la" ] FOR READ

...

PUBLIC SUB Process_Write(Data AS String)

  Content = Content & Data
  PRINT Data

END
```

EVENT

See *Events declaration.*

Events declaration

EVENT *Identifier* **(** [*Parameter #1* [, *Parameter #2* ...]] **)** [**AS Boolean**]

This declares a class event. This event is raised by a function call. You can specify if the event handler returns a boolean value.

Example :

```
EVENT BeforeSend(Data AS String) AS Boolean

...

' Raises the event

IF BeforeSend("MyData") THEN
  PRINT "Canceled !"
ENDIF
```

Exist

Boolean = **Exist (*Path*)**

Returns if a file or a directory exists.

Example :

```
PRINT Exist("/home/benoit/gambas")
⇨ True
PRINT Exist("/home/benoit/windows")
⇨ False
```

Exp

value = **Exp (*Number*)**

Computes the exponantial of a number.

Example :

```
PRINT Exp(1)
⇨ 2.718281828459
```

FALSE

The False constant. This constant is used to represent a false boolean value.

FINALLY

FINALLY

This instruction introduces the code executed at the end of the function, even if a error was raised during its execution.

The *finally* part is not mandatory. If there is a *catch* part in the function, the *finally* part must precede it.

If an error is raised during the execution of the *finally* part, it is normally propagated.

See *CATCH* for more details and for a code example.

Fix

Value = **Fix (*Number*)**

Returns the integer part of a number.

Example :

```
PRINT Fix(Pi)
⇨ 3
```

```
PRINT Fix(-Pi)
⇨ -3
```

FLUSH

FLUSH [*#File*]

Flushes a buffered stream. If no stream is specified, every opened streams are flushed.

FOR

FOR *Variable* = *Expression* **TO** *Expression* [**STEP** *Expression*]
  ...
NEXT

Repeats a loop while incrementing a variable. Note that the variable must be :

- Numeric, i.e. a byte, a short, an integer or a floating point number.
- A local variable.

Example :

```
DIM I AS Integer

FOR I = 1 TO 20 STEP 3
  PRINT I;
NEXT
PRINT

⇨ 1 4 7 10 13 16 19
```

FOR EACH

FOR EACH *Variable* **IN** *Expression*
  ...
NEXT

Repeats a loop while enumerating an object. *Expression* must be a reference to an enumerable object : for example, a collection, or an array. The order of the enumeration in not necessarily predictable.

Example :

```
DIM Dict AS NEW Collection

Dict["Blue"] = 3
Dict["Red"] = 1
Dict["Green"] = 2

FOR EACH Element IN Dict
  PRINT Element;
NEXT

⇨ 3 1 2
```

FOR EACH *Expression*
  ...
NEXT

This syntax must be used when *Expression* is a enumerable object that is not a container : for example, the result of a database query.

Example :

```
DIM Res AS Result

Res = DB.Exec("SELECT * FROM MyTable")

FOR EACH Res
  PRINT Res!Code; " "; Res!Name
NEXT
```

Format$

String = **Format$** (*Expression* [, *Format*])

Converts an expression to a string by using a format that depends on the type of the expression. Format can be a predefined format (an integer constant) or a user-defined format (a string that depicts the format).

This function uses localization information to format dates, times and numbers.

See *Predefined constants* for a list of predefined formats. If *Format* is not specified, *gb.Standard* is used.

A user-defined number format is described by the following characters :

- "+" prints the sign of the number.
- "-" prints the sign of the number only if it is negative.
- "#" prints a digit only if necessary.
- "0" always prints a digit, padding with a zero if necessary.
- "." prints the decimal separator.
- "%" multiplies the number by 100 and prints a per-cent sign.
- "E" introduces the exponential part of a float number. The sign of the exponent is always printed.

A user-defined date format is described by the following characters :

- "yy" prints the year on two digits.
- "yyyy" prints the year on four digits.
- "m" prints the month.
- "mm" prints the month on two digits.
- "mmm" prints the month in an abbreviatted string form.
- "mmmm" prints the month in its full string form.
- "d" prints the day.
- "dd" prints the day on two digits.
- "ddd" prints the week day in an abbreviated form.
- "dddd" prints the week day in its full form.
- "/" prints the date separator.
- "h" prints the hour.
- "hh" prints the hour on two digits.
- "n" prints the minutes.
- "nn" prints the minutes on two digits.
- "s" prints the seconds.
- "ss" prints the seconds on two digits.
- ":" prints the time separator.

User-defined numeric format examples :

```
PRINT Format$(Pi, "-#.###")
⇨ 3.142
PRINT Format$(Pi, "+0#.###0")
⇨ +03.1416
PRINT Format$(Pi / 10, "###.# %")
⇨ 31.4 %
PRINT Format$(-11 ^ 11, "#.##E##")
⇨ -2.85E+11
```

User-defined date and time format examples :

```
PRINT Format$(Now, "mm/dd/yyyy hh:mm:ss")
⇨ 04/15/2002 09:05:36
PRINT Format$(Now, "m/d/yy h:m:s")
⇨ 4/15/02 9:5:36
PRINT Format$(Now, "ddd dd mmm yyyy")
⇨ Mon Apr 15 2002
PRINT Format$(Now, "dddd dd mmmm yyyy")
⇨ Monday April 15 2002
```

Frac

Value = **Frac (** *Number* **)**

Computes the fractional part of a number.

Example :

```
PRINT Frac(Pi)
⇨ 0.14159265359
```

FUNCTION

See *Class method declaration*.

GOTO

GOTO *Label*

Jumps to a label declared elsewhere in the function.

Hex$

String = **Hex$ (** *Number* [**,** *Digits*] **)**

Gets the hexadecimal representation of a number. If *Digits* is specified, the representation is padded with unnecessary zeros so that *Digits* digits are returned.

Example :

```
PRINT Hex$(1972)
⇨ 7B4
PRINT Bin$(72, 8)
⇨ 000007B4
```

Hour

Result = **Hour (** *Date/Time* **)**

Returns the hours of a date/time value.

Example :

```
PRINT Now; " -> "; Hour(Now)
⇨ 05/16/2002 22:31:30 -> 22
```

IF

IF *Expression* **THEN**
    ...
[**ELSE IF** *Expression* **THEN**
    ...]
[**ELSE**
    ...]
ENDIF

Conditional control structure.

If / IIf

Value = **If (** *Boolean* **,** *True value* [**,** *False value*] **)**

Evaluate the *Boolean* expression, and return *True value* if this expression is true, or *False value* if this expression is false.

If *False value* is not specified, it is assumed to be NULL.

Be careful : Contrary to IF, both *True value* and *False value* are evaluated, whatever the value of *Boolean* is.

Example :

```
X = 7

PRINT If((X MOD 2) = 1, "Impair", "Pair")
⇨ Impair

PRINT If((X MOD 2) = 1, "Impair", 1 / 0)
** Division by zero **
```

INC

INC *Variable*

Increments a variable. *Variable* can be any target of an assignment, but must be numeric.

Example :

```
' First example

X = 7

INC X

PRINT X
⇨ 8

' Second example

DIM A[3, 3] AS FLOAT

X = 2
Y = 1

A[X, Y] = PI

INC A[X, Y]

PRINT A[X, Y]
⇨ 4.14159265359
```

INPUT

INPUT *Variable* [**,** *Variable* ...]

Reads the standard input, and converts elements separated by space characters or newlines with the **Val()** function before putting them into the variables.

INPUT *#File* , *Variable* [**,** *Variable* ...]

Same as above, except that the data are read from the stream *File*.

InStr

Position = **InStr (** *String* **,** *Substring* [**,** *Start*])

Returns the position of the first occurrence of *Substring* in *String*. If *Start* is specified, the search begins at the position *Start*.

If the substring is not found, InStr() returns zero.

Example :

```
PRINT Instr("Gambas is basic", "bas")
⇨ 4
PRINT Instr("Gambas is basic", "bas", 5)
⇨ 11
PRINT Instr("Gambas is basic", "not")
```

```
⇨ 0
```

Int

Value = **Int (** *Number* **)**

Returns the mathematical integer part of a number, i.e. the greater integer smaller than this number.

Example :

```
PRINT Int(Pi)
⇨ 3
PRINT Int(-Pi)
⇨ -4
```

IsBoolean / Boolean?

Boolean = **IsBoolean** (*Expression*)
Boolean = **Boolean?** (*Expression*)

Returns if an expression is a boolean.

Example :

```
PRINT IsBoolean(FALSE)
⇨ TRUE
PRINT IsBoolean(-1)
⇨ FALSE
PRINT IsBoolean(NULL)
⇨ FALSE
```

IsByte / Byte?

Boolean = **IsByte** (*Expression*)
Boolean = **Byte?** (*Expression*)

Return if an expression is a byte integer.

Example :

```
PRINT IsByte(1)
⇨ FALSE
PRINT IsByte(CByte(1))
⇨ TRUE
```

IsDate / Date?

Boolean = **IsDate** (*Expression*)
Boolean = **Date?** (*Expression*)

Returns if an expression is a date and time value.

Example :

```
PRINT IsDate(1)
⇨ FALSE
PRINT IsByte(CByte(1))
⇨ TRUE
```

IsFloat / Float?

Boolean = **IsFloat** (*Expression*)
Boolean = **Float?** (*Expression*)

Returns if an expression is a floating point number.

Example :

```
PRINT IsFloat(1)
⇨ FALSE
PRINT IsFloat(Pi)
⇨ TRUE
```

IsInteger / Integer?

Boolean = **IsInteger** (*Expression*)
Boolean = **Integer?** (*Expression*)

Returns if an expression is an integer.

Example :

```
PRINT IsInteger(1)
⇨ TRUE
PRINT IsInteger(Pi)
⇨ FALSE
```

IsNull / Null?

Boolean = **IsNull** (*Expression*)
Boolean = **Null?** (*Expression*)

Returns if an expression is null, i.e. if it is :

- The NULL constant.
- A null object reference.
- A zero length string.

- A null date.
- A uninitialized variant.

Example :

```
PRINT IsNull(NULL)
⇨ TRUE
PRINT IsNull("Gambas")
⇨ FALSE
PRINT IsNull("")
⇨ TRUE
```

IsNumber / Number?

Boolean = **IsNumber** (*Expression*)
Boolean = **Number?** (*Expression*)

Returns if an expression is a number, i.e. if it is :

- A boolean.
- Any integer number.
- A floating point number.

Example :

```
PRINT IsNumber(1)
⇨ TRUE
PRINT IsNumber(TRUE)
⇨ TRUE
PRINT IsNumber(Pi)
⇨ TRUE
PRINT IsNumber(Now)
⇨ FALSE
PRINT IsNumber(NULL)
⇨ FALSE
```

IsObject / Object?

Boolean = **IsObject** (*Expression*)
Boolean = **Object?** (*Expression*)

Returns if an expression is an object or a null reference.

Example :

```
DIM hRef AS Collection

hRef = NEW Collection

PRINT IsObject(hRef)
⇨ TRUE
PRINT IsObject(NULL)
⇨ TRUE
PRINT IsObject(Pi)
⇨ FALSE
```

IsShort / Short?

Boolean = **IsShort** (*Expression*)
Boolean = **Short?** (*Expression*)

Returns if an expression is a short integer.

Example :

```
PRINT IsShort(1)
⇨ FALSE
PRINT IsShort(CShort(1))
⇨ TRUE
```

IsString / String?

Boolean = **IsString** (*Expression*)
Boolean = **String?** (*Expression*)

Returns if an expression is a string.

Example :

```
PRINT IsString("Gambas")
⇨ TRUE
PRINT IsString("")
⇨ TRUE
PRINT IsString(NULL)
⇨ TRUE
PRINT IsString(Pi)
⇨ FALSE
```

KILL

KILL *Path*

Removes a file.

Labels

Identifier :

A label indicates a target for the GOTO instruction.

A label is local to a function or procedure.

LAST

Returns a reference to the last object that raised an event.

Example :

```
DIM hButton[3] AS Button

hButton[0] = NEW Button AS "MyButtons"
hButton[0].Text = "Red"

hButton[1] = NEW Button AS "MyButtons"
hButton[1].Text = "Green"

hButton[2] = NEW Button AS "MyButtons"
hButton[2].Text = "Blue"

...

PUBLIC SUB MyButtons_Click()

  PRINT LAST.Text

END
```

Left$

Result = **Left$ (** *String* [**,** *Length*])

Returns the *Length* first characters of a string. If *Length* is not specified, the first character of the string is returned. If *Length* is negative, all the string except the (-*Length*) last characters is returned.

Example :

```
PRINT Left$("Gambas", 4)
⇨ Gamb
PRINT Left$("Gambas")
⇨ G
PRINT Left$("Gambas", -1)
⇨ Gamba
```

Len

Length = **Len (** *String* **)**

Returns the length of a string.

Example :

```
PRINT Len("Gambas")
⇨ 6
PRINT Len("")
⇨ 0
```

LTrim$

Result = **LTrim$ (** *String* **)**

Strips white spaces from the left of a string. A white space is any character whose ASCII code is stricly lower than 32.

Example :

```
PRINT "<"; LTrim$(" Gambas"); ">"
⇨ <Gambas>
PRINT "<"; LTrim$(" Gambas  "); ">"
⇨ <Gambas  >
```

LIKE

Boolean = *String* **LIKE** *Pattern*

Returns TRUE if *String* matches *Pattern*. The pattern can contain the following generic characters :

- "*" matches any number of any character.
- "?" matches any single character.
- "[x-y]" matches any character in the interval.
- "[^x-y]" matches any character not in the interval.
- "\" prevents a character to be interpreted as generic.

Example :

```
PRINT "Gambas" LIKE "G*"
⇨ TRUE
PRINT "Gambas" LIKE "?[Aa]*"
⇨ TRUE
PRINT "Gambas" LIKE "G[Aa]\*"
⇨ FALSE
PRINT "Gambas" LIKE "G[^Aa]*"
⇨ FALSE
```

LINE INPUT

LINE INPUT *Variable*

Reads an entire line of text on the standard input.

LINE INPUT #*File* **,** *Variable*

Same as above, except that the data are read from the stream *File*.

Example :

```
' Print a file to standard output

OPEN FileName FOR READ AS #hFile

WHILE NOT Eof(hFile)
  LINE INPUT #hFile, OneLine
```

```
    PRINT OneLine
  WEND

  CLOSE #hFile
```

Local variable declaration

[**DIM**] *Identifier* **AS** *Datatype*

Declare a local variable in a procedure or function. This variable is only accessible to the procedure or function where it is declared.

Example :

```
DIM Val AS INTEGER
DIM Name AS STRING
DIM Matrix[3, 3] AS FLOAT
DIM AnObject AS OBJECT
...
```

Lof

Length = **Lof (** *File* **)**

Returns the length of a opened stream.

Example :

```
OPEN FileName FOR READ AS #hFile
...
PRINT "File length is"; Lof(hFile)
```

Log

value = **Log (** *Number* **)**

Computes the neperian logarithm of a number.

Example :

```
PRINT Log(2.71828)
⇨ 0.999999327347
PRINT Log(1)
⇨ 0
```

Log10

value = **Log10 (** *Number* **)**

Computes the decimal logarithm of a number. Log10(x) = Log(x) / Log(10).

Example :

```
PRINT Log10(10)
```

```
⇨ 1
```

LOOP

LOOP [**UNTIL** *Expression*]

Ends a loop structure delimited by DO ... LOOP instructions. If UNTIL is specified, the loop is stopped when *Expression* is true.

Lower$ / LCase $

Result = **Lower$ (** *String* **)**
Result = **LCase$ (** *String* **)**

Returns a string converted to lower case.

Example :

```
PRINT Lower$("Gambas ALMOST Means BASIC !")
⇨ gambas almost means basic !
```

Max

value = **Max (** *Expression* [, *Expression* ...] **)**

Returns the greater expression of the list. Expressions must be numbers or date/time values.

Example :

```
PRINT Max(6, 4, 7, 1, 3)
⇨ 7
PRINT Max(Now, CDate("01/01/1900"), CDate("01/01/2100"))
⇨ 01/01/2100
```

ME

ME

Returns a reference to the current object.

ME is mandatory when you want to call an inherited method, or access an inherited property or variable.

Example :

```
' Gambas form

...

PUBLIC SUB SetTitle(Title AS String)

  ME.Text = Title
```

```
END

PUBLIC SUB MoveRight(Step AS Integer)

  ME.Move(ME.X + Step, ME.Y)

END
```

Methods declaration

[**STATIC**] (**PUBLIC** | **PRIVATE**) (**PROCEDURE** | **SUB** | **FUNCTION**) *Identifier*
  ([[**OPTIONAL**] *Parameter #1* [, [**OPTIONAL**] *Parameter #2* ...] **)**
  [**AS** *Datatype*]
 ...
END

This declares a class method. The END keyword indicates the end of the method.

- This method is accessible everywhere in the class it is declared.
- If the PUBLIC keyword is specified, it is also accessible to the other classes having a reference to an object of this class.
- If the STATIC keyword is specified, the method can only access to the static variables of the class.
- If the keyword FUNCTION is specified, the method is a function and the return datatype must be specified.
- If the keyword OPTIONAL is specified, the corresponding parameter is optional. Then you can specify a default value after the parameter declaration by using the equal sign.

Example :

```
STATIC PUBLIC PROCEDURE Main()
...
PUBLIC FUNCTION Calc(A AS Float, B AS Float) AS Float
...
PRIVATE SUB DoIt(Command AS String, OPTIONAL SaveIt AS BOOLEAN = TRUE)
...
```

Mid$

Result = **Mid$ (** *String* **,** *Start* [**,** *Length*] **)**

Returns a substring containing the *Length* characters from the position *Start*. If *Length* is not specified, everything from the position *Start* is returned. If *Length* is negative, everything from the position *Start* except the (-*Length*) last characters is returned.

Example :

```
PRINT Mid$("Gambas", 3, 2)
⇨ mb
PRINT Mid$("Gambas", 4)
⇨ bas
PRINT Mid$("Gambas", 2, -1)
⇨ amba
```

Min

value = **Min (** *Expression* [, *Expression* ...] **)**

Returns the smaller expression of the list. Expressions must be numbers or date/time values.

Example :

```
PRINT Min(6, 4, 7, 1, 3)
⇨ 1
PRINT Min(Now, CDate("01/01/1900"), CDate("01/01/2100"))
⇨ 01/01/1900
```

Minute

Result = **Minute (** *Date/Time* **)**

Returns the minutes of a date/time value.

Example :

```
PRINT Now; " -> "; Hour(Now)
⇨ 05/16/2002 22:31:30 ->  31
```

MKDIR

MKDIR *Path*

Creates a directory.

MOD

See *Arithmetic operators*.

Month

Result = **Month (** *Date/Time* **)**

Returns the month component of a date/time value.

Example :

```
PRINT Now; " -> "; Month(Now)
⇨ 05/16/2002 22:31:30 ->  5
```

NEXT

See *FOR* and *FOR EACH*.

NEW

Variable = **NEW** *Class* [(*constructor parameters...*)] [**AS** *Name*]

Instanciates the class *Class*. The object or class where the object is instanciated is its *parent*. If a name is specified, the new object will be able to raise events by calling a public procedure or function in its parent. The name of this event handler is the name of the object followed by an underscore and the name of the event.

Note : NEW is not an operator. You can only use it within an assignment.

Example :

```
hButton = NEW Button AS "MyButton"
...
PUBLIC PROCEDURE MyButton_Click()
  PRINT "My button was clicked !"
END
```

Two differents object can have the same event name. Thus, you can manage events of multiple objects in the same event procedure, provided these objects raise the same events.

New

Object = **New (** *Class* **,** [*constructor parameters...*] **)**

Instanciates the class *Class*. This routine works exactly like the NEW operator, except that the class name is specified at runtime and not at compile time.

Example :

```
hButton = New("Button", hParent)
```

has exactly the same effect as :

```
hButton = NEW Button(hParent)
```

NOT

Result = **NOT** *Expression*

Computes the logical *not* of an expression. If *Expression* is a string or an object, it returns True if *Expression* is null. See *NULL* or *IsNull* for more detail.

Example :

```
PRINT NOT TRUE
```

```
⇨ False
PRINT NOT FALSE
⇨ True
PRINT NOT 11
⇨ -12
PRINT NOT CByte(11)
⇨ 244
PRINT NOT "Gambas"
⇨ False
PRINT NOT ""
⇨ True
```

Now

Time = **Now ( )**

Returns the current date and time.

Example :

```
PRINT Now
⇨ 05/16/2002 22:31:30
```

NULL

NULL

The null constant. This constant is used to represent a null object reference, a zero length string, a null date, or an uninitialized variant.

OPEN

OPEN *File name* **FOR** (**READ** | **WRITE** | **CREATE** | **APPEND**) [**DIRECT**] **AS** #*Variable*

Opens a file for reading, writing, creating or appending data. If the **DIRECT** keyword is specified, the input-output are not buffered.

The variable receives the object that represents the opened stream.

OR

Result = *Expression* **OR** *Expression*

Computes the logical *or* of two expressions.

Example :

```
PRINT TRUE OR FALSE
⇨ True
PRINT FALSE OR FALSE
⇨ False
```

```
PRINT 7 OR 11
⇨ 15
```

PI

Result = **Pi (** [*Number*] **)**

Returns π * *Number*. If *Number* is not specified, it is assumed to be one.

Example :

```
PRINT Pi
⇨  3.14159265359
PRINT Pi(0.5)
⇨  1.570796326795
```

Predefined classes

Application	Global information about application.
Class	A class loaded by the interpreter.
Classes	A collection of all classes loaded by the interpreter.
Collection	An hash table whose keys are string and values are *Variants*.
Date[]	An array of *Dates*.
Error	Error management.
File	File management.
Float[]	An array of *Floats*.
Integer[]	An array of *Integers*.
Libraries	A collection of all components loaded by the interpereter
Library	A component loaded by the interpreter.
Object	Object management.
Object[]	An array of *Objects*.
Process	Process management.
String[]	An array of Strings.
Symbol	A class symbol.
Variant[]	An array of *Variants*.

Note that arrays are implemented as native classes.

See also *Datatypes*.

Predefined constants

Datatypes

`gb.Null`	Null value
`gb.Boolean`	Boolean value
`gb.Byte`	Byte integer number
`gb.Short`	Short integer number
`gb.Integer`	Integer number
`gb.Float`	Floating point number
`gb.Date`	Date and time value
`gb.String`	Character string
`gb.Variant`	Variant
`gb.Object`	Object reference

File types

`gb.File`	Regular file
`gb.Directory`	Directory
`gb.Device`	Special file for a device
`gb.Pipe`	Named pipe
`gb.Socket`	Special file for a socket
`gb.Link`	Symbolic link

String constants

`gb.NewLine`	Newline character. Equivalent to Chr$(13)
`gb.Tab`	Tab character. Equivalent to Chr$(9)

Sort types

`gb.Binary`	Binary sort
`gb.Case`	Case insensitive sort
`gb.Lang`	Language based sort

Predefined numeric formats

`gb.GeneralNumber`	Write a number with twelve decimal digits. Use scientific format if its absolute value is lower than 10^{-4} or greater than 10^{7}.
`gb.Fixed`	Equivalent to "0.00"
`gb.Percent`	Equivalent to "###%"
`gb.Scientific`	Write a number with its exponent and eighteen decimal digits.

Predefined date and time formats

`gb.GeneralDate`	Write a date only if the date and time value has a date part, and write a time only if it has a date part.
`gb.LongDate`	Long date format.
`gb.MediumDate`	Medium date format.
`gb.ShortDate`	Short date format.
`gb.LongTime`	Long time format.
`gb.MediumTime`	Medium time format.
`gb.ShortTime`	Short time format.

Miscellaneous formats

`gb.Standard`	Use `gb.GeneralNumber` for formatting numbers and `gb.GeneralDate` for formatting dates and times.

PRINT

PRINT *Expression* [(; | ,) *Expression* ...] [(; | ,)]

Prints expressions to the standard output. The expressions are converted to strings by the Str() function.

If there is no semi-colon nor comma after the last expression, a newline character is printed after the last expression.

If a comma is used instead of a semi-colon, then a tab character (ASCII code 9) is printed to separate the expressions.

PRINT #*File* **,** *Expression* [(; | ,) *Expression* ...] [(; | ,)]

Same as above, except that expressions are sent to the stream *File*.

PRIVATE

See *Class Variable declaration* and *Class method declaration*.

PROCEDURE

See *Method declaration*.

PUBLIC

See *Variable declaration* and *Method declaration*.

Randomize

Randomize ( )

Initializes the pseudo-random numbers generator.

Rad

Value = **Rad (** *Angle* **)**

Converts degrees to radians.

Example :

```
PRINT Rad(90)
⇨  1.570796326795
PRINT Rad(180) - Pi
```

```
⇨ 0
```

READ

READ *Variable* [, *Length*]

Reads the standard output as binary data whose type is given by the type of the variable. The binary representation is the one used by the **WRITE** instruction.

If *Variable* is a string, you can specify a length that indicates the number of bytes to read. If no length is specified for a string, it is is read from the stream.

READ *#File* , *Variable* [, *Length*]

Same as above, except that the data are read from the stream File.

RENAME

RENAME *Old name* **AS** *New name*

Renames or move a file or a directory.

REPEAT

REPEAT

Begins a loop structure delimited by REPEAT ... UNTIL instructions.

Replace$

Result = **Replace$ (** *String* , *Pattern* , *ReplaceString*)

Replaces every occurrence of the string *Pattern* in the string *String* by the string *ReplaceString*, and return the result.

If *String* is null, then a null string is returned.

If *Pattern* is null, then the string *String* is returned.

Example :

```
PRINT Replace$("Gambas is basic", "bas", "BAS")
⇨ GamBAS is BASic
PRINT Replace$("Gambas is basic", "a", "")
⇨ Gmbs is bsic
PRINT Replace$("Gambas is basic", " ", "--")
⇨ Gambas--is--basic
```

RETURN

RETURN [*Expression*]

Quits a procedure or a function by returning the value of *Expression*.

Right$

Result = **Right$ (** *String* [**,** *Length*])

Returns the *Length* last characters of a string. If *Length* is not specified, the last character of the string is returned. If *Length* is negative, all the string except the (-*Length*) first characters is returned.

Example :

```
PRINT Right$("Gambas", 4)
⇨ mbas
PRINT Right$("Gambas")
⇨ s
PRINT Right$("Gambas", -1)
⇨ ambas
```

RInStr

Position = **RInStr (** *String* **,** *Substring* [**,** *Start*])

Returns the position of the last occurrence of *Substring* in *String*. If *Start* is specified, the search stops at the position *Start*.

If the substring is not found, RInStr() returns zero.

Example :

```
PRINT RInstr("Gambas is basic", "bas")
⇨ 11
PRINT RInstr("Gambas is basic", "not")
⇨ 0
```

RMDIR

RMDIR *Path*

Removes a directory. The directory must be empty.

Rnd

Rnd ([*Min* [**,** *Max*] **)**

Computes a pseudo-random floating point number, using the *Lehmer* algorithm.

- If no parameters is specified, return a pseudo-random number in the interval [0 , 1 [.
- If the first parameter is specified, return a pseudo-random in the interval [0 , *Min* [.
- If the two parameters are specified, return a pseudo-random in the interval [*Min* , *Max* [.

Example :

```
PRINT Rnd
⇨  0.019539254718
PRINT Rnd(2)
⇨  0.040205506608
PRINT Rnd(Pi, Pi(2))
⇨  3.204108046818
```

Rol

Value = **Rol (** *Number* **,** *Bits* **)**

Returns *Number*, *Bits* bits left rotated.

Ror

Value = **Ror (** *Number* **,** *Bits* **)**

Returns *Number*, *Bits* bits right rotated.

Round

Value = **Round (** *Number* [**,** *Digits*] **)**

Rounds a number to its nearest integer if *Digits* is not specified.

If *Digits* is specified, rounds to 10 ^ (-*Digits*).

RTrim$

Result = **RTrim$ (** *String* **)**

Strips white spaces from the right of a string. A white space is any character whose ASCII code is lower or equal than 32.

Example :

```
PRINT "<"; RTrim$("Gambas  "); ">"
⇨ <Gambas>
PRINT "<"; RTrim$(" Gambas  "); ">"
⇨ < Gambas>
```

Second

Result = **Second (** *Date/Time* **)**

Returns the seconds of a date/time value.

Example :

```
PRINT Now; " -> "; Second(Now)
⇨ 05/16/2002 22:31:30 ->  30
```

Seek

Position = **Seek (** *File* **)**

Returns the current read/write stream pointer of the stream *File*.

SEEK

SEEK #*File* **,** *Position*

Defines the position of the stream pointer, for the next read/write operation.

If *Position* is negative, then the stream pointer is moved relatively to the end of the file.

To move the stream pointer after the end of the file, you must use the **Lof()** function.

Example :

```
' Move to the beginning of the file
SEEK #hFile, 0
' Move after the end of the file
SEEK #hFile, Lof(#hFile)
' Move 100 bytes before the end of the file
SEEK #hFile, -100
```

SELECT

SELECT *Expression*
[**CASE** *Expression* [**,** *Expression* ...]
...]
[**CASE** *Expression* [**,** *Expression* ...]
...]
[(**CASE ELSE** | **DEFAULT**)
...]
END SELECT

Selects an expression to compare, and execute the code enclosed in the corresponding matching CASE statement. If no CASE statement matches, the DEFAULT or CASE ELSE statement is executed.

Sgn

Sign = **Sgn (** *Number* **)**

Returns the sign of a number.

- If the number is zero, it returns zero.
- If the number is strictly positive, it returns the integer number +1.
- If the number is strictly negative, it returns the integer number -1.

Example :

```
PRINT Sgn(Pi)
⇨  1
PRINT Sgn(-Pi)
⇨ -1
PRINT Sgn(0)
⇨  0
```

SHELL

SHELL *Command* [**WAIT**] [**FOR** (**READ** | **WRITE** | **READ WRITE**) [**AS** *Variable*]

Executes a command via a system shell. An internal *Process* object is created to manage the command.

If WAIT is specified, then the interpreter waits for the command ending. Otherwise, the command is executed in background.

If FOR is specified, then the command input-outputs are redirected so that your program intercepts them :

- If WRITE is specified, you can send data to the command standard input via a method of the class Process. Note that you need a reference to the Process object for that.
- If READ is specified, then events will be generated each time the command sends data to its standard output streams : the event *Write* is raised when data are sent to the standard output stream, and the event *Error* is raised when data are sent to the standard error stream.

Lastly, you can get a reference to the internal Process object into a variable *Variable* by specified the keyword AS.

Example :

```
' Get the content of a directory

SHELL "ls -la > /tmp/result" WAIT
Content = File.Load("/tmp/result")

' Same thing, but in background

SHELL "ls -la > /tmp/result" FOR READ

...

PUBLIC SUB Process_Write(Data AS String)

  Content = Content & Data

END
```

Shl

Value = **Shl** (*Number* , *Bits*)

Returns *Number*, *Bits* bits left shifted.

Example :

```
PRINT Shl(11, 3)
⇨  88
```

Shr

Value = **Shr** (*Number* , *Bits*)

Returns *Number*, *Bits* bits right shifted.

Example :

```
PRINT Shr(11, 3)
⇨  1
```

Sin

Value = **Sin** (*Angle*)

Computes the sine of an angle. The angle is specified in radians.

Example :

```
PRINT Sin(Pi/2)
⇨  1
```

Sinh

Value = **Sinh** (*Number*)

Computes the hyperbolic sine of a number.

Example :

```
PRINT Sinh(1)
⇨  1.175201193644
```

Space$

String = **Space$** (*Length*)

Returns a string containing *Length* spaces.

Example :

```
PRINT "<"; Space$(8); ">"
⇨ <        >
```

Split

Array = **Split (** *String* [**,** *Separators* **,** *Escape*] **)**

Splits a string into substring delimited by *Separators*. *Escape* characters can be specified : any separator characters enclosed between two escape characters are ignored in the splitting process.

By default, the comma character is the separator, and there are no escape characters.

This function returns an array filled with every detected substring.

Example :

```
DIM Elt AS String[]
DIM Sub AS String

Elt = Split("Gambas Almost Means BASIC ! 'agree ?'", " ", "'")

FOR EACH Sub IN Elt
  PRINT Sub
NEXT

⇨
Gambas
Almost
Means
BASIC
!
agree ?
```

Sqr

value = **Sqr (** *Number* **)**

Computes the square root of a number.

Example :

```
PRINT Sqr(2)
⇨  1.414213562373
```

Stat

Type = **Stat (** *Path* **)**

Returns the type of a file or a directory. The global static variables of the class *File* are also initialized with information about the file : size, last modification time... See *Predefined Constants* and the class *File* for more details.

Example :

```
PRINT Stat("/home") = gb.Directory
```

```
⇨ True
```

STATIC

See *Class Variable declaration* and *Class method declaration*.

STEP

See *FOR*.

Str$

String = **Str$** (*Expression*)

Converts an expression into its printable string representation. It is the exact contrary of **Val** ( ).

The current localization is used to convert numbers and dates.

Example :

```
' Print on standard output or in a message

PUBLIC CONST ON_STDOUT AS Integer = 1
PUBLIC CONST ON_MESSAGE AS Integer = 2

SUB PrintOn(Where AS Integer, What AS Variant)

  IF Where = ON_STDOUT THEN
    PRINT What
  ELSE IF Where = ON_MESSAGE THEN
    Message(Str$(What))
  ENDIF

END
```

String operators

String **&** *String*	Concatenate two strings.
String **&/** *String*	Concatenate two strings that contain file names. Add a path separator between the two strings if necessary.
String **LIKE** *Pattern*	Pattern matching. See *LIKE*.
String = *String*	Are the two strings equal ?
String <> *String*	Are the two strings different ?
String < *String*	Is the first string lower than the second ?
String > *String*	Is the first string greater than the second ?
String <= *String*	Is the first string lower or equal than the second ?
String >= *String*	Is the first string greater or equal than the second ?

Note : all comparisons are case sensitive.

String$

String = **String$ (** *Length* **,** *Pattern* **)**

Returns a string containing *Length* times the *Pattern*.

Example :

```
PRINT String$(12, "*")
⇨ ************
PRINT String$(2, "Gambas")
⇨ GambasGambas
```

SUB

See *Class method declaration*.

Tan

value = **Tan** (*angle*)

Computes the tangent of an angle. The angle is specified in radians.

Example :

```
PRINT Tan(Pi/4)
⇨  1
```

Tanh

Value = **Tanh** (*Number*)

Computes the hyperbolic tangent of a number.

Example :

```
PRINT Tanh(1)
⇨  0.655794202633
```

Temp / Temp$

File name = **Temp$** ( )

Returns a unique file name useful to create temporary files.

Example :

```
PRINT Temp$()
```

```
⇨ /tmp/gambas/14593.1.tmp
PRINT Temp$()
⇨ /tmp/gambas/14593.2.tmp
```

THEN

See *IF*.

Time

Result = **Time (** *Date/Time* **)**

Returns the time part of a date/time value.

Example :

```
PRINT Time(Now)
⇨ 14:08:25
```

Date = Time **(** *Hours* **,** *Minutes* **,** *Seconds* **)**

Makes a time from its components.

Example :

```
PRINT Time(14, 08, 25)
⇨ 14:08:25
```

Timer

Time = **Timer ( )**

Returns the number of elapsed seconds from the beginning of the program.

Example :

```
PRINT Timer
⇨  0.291657006775
```

TO

See *FOR*.

Trim$

Result = **Trim$** (*String*)

Strips white spaces from a string. A white space is any character whose ASCII code is stricly lower than 32.

Example :

```
PRINT "<"; Trim$("Gambas"); ">"
⇨ <Gambas>
PRINT "<"; Trim$("  \nGambas " & Chr$(9) & " "); ">"
⇨ <Gambas>
```

TRUE

The True constant. This constant is used to represent a true boolean value.

TRY

TRY *Statement*

Tries to execute a statement, without raising an error.

Example :

```
' Remove a file even if it exists
TRY KILL FileName
' Test if it has succeeded
IF ERROR THEN PRINT "Cannot remove file"
```

TypeOf

Type = **TypeOf** (*Expression*)

Returns the type of an expression as an integer value.

See *Predefined constants* for a list of the datatypes returned by this function.

UNTIL

UNTIL *Expression*

Ends a loop structure delimited by REPEAT ... UNTIL instructions. The loop is repeated until *Expression* is true.

Upper$ / UCase $

Result = **Upper$ (** *String* **)**
Result = **UCase$ (** *String* **)**

Returns a string converted to upper case.

Example :

```
PRINT Upper$("Gambas ALMOST Means BASIC !")
⇨ GAMBAS ALMOST MEANS BASIC !
```

Val

Expression = **Val** (*String*)

Converts a string into a boolean, a number or a date, according to the content of the string. The current localization is used to convert numbers and dates.

The conversion algorithm is the following :

- If the string can be interpreted as a date & time (with date or time separators), then the date & time is returned.
- Else, if the string can be interpreted as a floating point number, then this floating point number is returned.
- Else, if the string can be interpreted as a integer number, then this integer number is returned.
- Else, if the string is "`True`" or "`False`", then the matching boolean value is returned.
- Otherwise, `Null` is returned.

Example :

```
PRINT Val("09/06/72 01:00")
⇨ 09/06/72 01:00:00
PRINT Val("3.1415")
⇨  3.1415
PRINT Val("-25")
⇨ -25
PRINT Val("True")
⇨ True
PRINT IsNull(Val("Gambas"))
⇨ True
```

Variable declarations

[**STATIC**] (**PUBLIC** | **PRIVATE**) *Identifier* [*Array declaration*] **AS** [**NEW**] *Datatype*

This declares a class global variable.

- This variable is accessible everywhere in the class it is declared.
- If the PUBLIC keyword is specified, it is also accessible to the other classes having a reference to an object of this class.
- If the STATIC keyword is specified, the same variable will be shared with every object of this class.

- If the NEW keyword is specified, the variable is initialized with a new instance of the class specified with *Datatype*.

Example :

```
STATIC PUBLIC GridX AS Integer
STATIC PRIVATE bGrid AS Boolean

PUBLIC Name AS String
PUBLIC Control AS Object

STATIC PRIVATE Synonymous AS NEW Collection

PRIVATE hHandle[8] AS Label
```

WAIT

WAIT [*Delay*]

Calls the event loop. If *Delay* is specified, does not return until *Delay* seconds elaps.

WeekDay

Result = **WeekDay (** *Date/Time* **)**

Returns the week day of a date/time value. See *Predefined constants* for a list of constants associated with week days.

Example :

```
PRINT Now; " -> "; WeekDay(Now)
⇨ 05/16/2002 22:31:30 ->  4
```

WHILE

WHILE *Expression*

Begins a loop structure delimited by WHILE ... WEND instructions. The loop is repeated while *Expression* is true.

WEND

WEND

Ends a loop structure delimited by WHILE ... WEND instructions.

WITH

WITH *Expression*
  ...
END WITH

Between the WITH and the END WITH instruction, an expression beginning with a point is referring to *Expression.* So *Expression* must be an object.

Example :

```
WITH hButton
  .Text = "Cancel"
END WITH
```

is equivalent to

```
hButton.Text = "Cancel"
```

WRITE

WRITE *Expression* [**,** *Length*]

Writes expressions to the standard output using their binary representation.

If *Expression* is a string, you can specify a length that indicates the number of bytes to write. If no length is specified for a string, it is directly written to the stream just before the string data.

WRITE #*File* **,** *Expression* [**,** *Length*]

Same as above, except that the expressions are sent to the stream *File*.

XOR

Result = *Expression* **XOR** *Expression*

Computes the logical *exclusive or* of two expressions.

Example :

```
PRINT TRUE XOR FALSE
⇨ True
PRINT TRUE XOR TRUE
⇨ False
PRINT 7 XOR 11
⇨ 12
```

Year

Result = **Year (** *Date/Time* **)**

Returns the year component of a date/time value.

Example :

```
PRINT Now; " -> "; Year(Now)
⇨ 05/16/2002 22:31:30 ->  2002
```