INDEX Funktionsbeschreibungen

Mathematical Procedures

ABS (return absolute value)

ABS(*expression***)**

ABS Returns absolute value.

expression A constant, variable, or expression.

The **ABS** procedure returns the absolute value of an *expression*. The absolute value of a number is always positive (or zero).

Return Data Type: REAL or DECIMAL

Example:

```
C = ABS(A - B)                !C is absolute value of the difference
IF B < 0 THEN B = ABS(B).     !If b is negative make it positive
```

See Also:

BCD Operations and Procedures

INRANGE (check number within range)

INRANGE(*expression*,*low*,*high***)**

INRANGE	Return number in valid range.
expression	A numeric constant, variable, or expression.
low	A numeric constant, variable, or expression of the lower boundary of the range.
high	A numeric constant, variable, or expression of the upper boundary of the range.

The **INRANGE** procedure compares a numeric *expression* to an inclusive range of numbers. If the value of the *expression* is within the range, the procedure returns the value 1 for "true." If the *expression* is greater than the *high* parameter, or less than the *low* parameter, the procedure returns a zero for "false."

Return Data Type: LONG

Example:

```
IF INRANGE(Date % 7,1,5)   !If this is a week day
  DO WeekdayRate           ! use the weekday rate
ELSE                       !Otherwise
  DO WeekendRate           ! use the weekend rate
END
```

INT (truncate fraction)

INT(*expression***)**

INT Return integer.

expression A numeric constant, variable, or expression.

The **INT** procedure returns the integer portion of a numeric expression. No rounding is performed, and the sign remains unchanged.

Return Data Type: REAL or DECIMAL

Example:

```
   !INT(8.5)        returns 8
   !INT(-5.9)       returns -5

x = INT(y)          !Return integer portion of y variable contents
```

See Also:

BCD Operations and Procedures

ROUND

LOGE (return natural logarithm)

LOGE(*expression***)**

LOGE Returns the natural logarithm.

expression A numeric constant, variable, or expression. If the value of the *expression* is less than zero, the return value is zero. The natural logarithm is undefined for values less than zero.

The **LOGE** (pronounced "log-e") procedure returns the natural logarithm of a numeric *expression*. The natural logarithm of a value is the power to which **e** must be raised to equal that value. The value of **e** used internally by the Clarion library for these calculations is 2.71828182846.

Return Data Type: REAL

Example:

```
 !LOGE(2.71828182846)  returns 1
 !LOGE(1)              returns 0

LogVal = LOGE(Val)     !Get the natural log of Val
```

See Also:

LOG10

LOG10 (return base 10 logarithm)

LOG10(*expression***)**

LOG10	Returns base 10 logarithm.
expression	A numeric constant, variable, or expression. If the value of the *expression* is zero or less, the return value will be zero. The base 10 logarithm is undefined for values less than or equal to zero.

The **LOG10** (pronounced "log ten") procedure returns the base 10 logarithm of a numeric *expression*. The base 10 logarithm of a value is the power to which 10 must be raised to equal that value.

Return Data Type: REAL

Example:

```
   !LOG10(10)     returns 1
   !LOG10(1)      returns 0

LogStore = LOG10(Var)       !Store the log 10 of var
```

See Also:

LOGE

RANDOM (return random number)

RANDOM(*low*,*high***)**

RANDOM Returns random integer.

low A numeric constant, variable, or expression for the lower boundary of the range.

high A numeric constant, variable, or expression for the upper boundary of the range.

The **RANDOM** procedure returns a random integer between the *low* and *high* values, inclusively. The *low* and *high* parameters may be any numeric expression, but only their integer portion is used for the inclusive range.

Return Data Type: LONG

Example:

```
Num                 BYTE,DIM(49)
LottoNbr            BYTE,DIM(6)
  CODE
  CLEAR(Num)
  CLEAR(LottoNbr)
  LOOP X# = 1 TO 6
    LottoNbr[X#] = RANDOM(1,49)       !Pick numbers for Lotto
    IF NOT Num[LottoNbr[X#]]
      Num[LottoNbr[X#]] = 1
    ELSE
      X# -= 1
  . .
```

ROUND (return rounded number)

ROUND(*expression*,*order***)**

ROUND Returns rounded value.

expression A numeric constant, variable, or expression.

order A numeric expression with a value equal to a power of ten, such as 1, 10, 100, 0.1, 0.001, etc. If the value is not an even power of ten, the next lowest power is used; 0.55 will use 0.1 and 155 will use 100.

The **ROUND** procedure returns the value of an *expression* rounded to a power of ten. If the *order* is a LONG or DECIMAL Base Type, then rounding is performed as a BCD operation. Note that if you want to round a real number larger than 1e30, you should use ROUND(num,1.0e0), and not ROUND(num,1). The ROUND procedure is very efficient ("cheap") as a BCD operation and should be used to compare REALs to DECIMALs at decimal width.

Return Data Type: DECIMAL or REAL

Example:

```
 !ROUND(5163,100)        returns 5200
 !ROUND(657.50,1)        returns 658
 !ROUND(51.63594,.01)    returns 51.64

Commission = ROUND(Price / Rate,.01)  !Round the commission to the nearest cent
```

See Also:

BCD Operations and Procedures

SQRT (return square root)

SQRT(*expression***)**

SQRT Returns square root.

expression A numeric constant, variable, or expression. If the value of the expression is less than zero, the return value is zero.

The **SQRT** procedure returns the square root of the *expression*. If X represents any positive real number, the square root of X is a number that, when multiplied by itself, produces a product equal to X.

Return Data Type: REAL

Example:

```
Length = SQRT(X^2 + Y^2) !Find the distance from 0,0 to x,y (pythagorean theorem)
```

Trigonometric Procedures

Trigonometric procedures return values representing angles and ratios of the sides of a right triangle (a triangle containing a 90-degree angle). The hypotenuse is the side of the triangle opposite the right (90-degree) angle. For either of the other two angles, the adjacent side forms the angle with the hypotenuse, and the opposite side is opposite the angle. (See any good Trigonometry text for further explanation of these terms.)

Angles are expressed in radians. PI is a constant which represents the ratio of the circumference and radius of a circle. There are 2*PI radians (or 360 degrees) in a circle.

The following equates provide high precision constants for PI and the conversion factors between degrees and radians.

```
PI EQUATE(3.1415926535898)        !The value of PI
Rad2Deg EQUATE(57.295779513082)   !Number of degrees in a radian
Deg2Rad EQUATE(.0174532925199)    !Number of radians in a degree
```

See Also:

- SIN (return sine)
- COS (return cosine)
- TAN (return tangent)
- ASIN (return arcsine)
- ACOS (return arccosine)
- ATAN (return arctangent)

SIN (return sine)

SIN(*radians***)**

SIN Returns sine.

radians A numeric constant, variable or expression for the angle expressed in radians.

The **SIN** procedure returns the trigonometric sine of an angle measured in *radians*. The sine is the ratio of the length of the angle's opposite side divided by the length of the hypotenuse.

Return Data Type: REAL

Example:

```
Angle = 45 * Deg2Rad        !Translate 45 degrees to Radians
SineAngle = SIN(Angle)      !Get the sine of 45 degree angle
```

See Also:

TAN

ATAN

ASIN

COS

ACOS

COS (return cosine)

COS(*radians***)**

COS Returns cosine.

radians A numeric constant, variable or expression for the angle in radians.

The **COS** procedure returns the trigonometric cosine of an angle measured in *radians*. The cosine is the ratio of the length of the angle's adjacent side divided by the length of the hypotenuse.

Return Data Type: REAL

Example:

```
Angle = 45 * Deg2Rad          !Translate 45 degrees to Radians
CosineAngle = COS(Angle)      !Get the cosine of 45 degree angle
```

See Also:

TAN

ATAN

SIN

ASIN

ACOS

TAN (return tangent)

TAN(*radians***)**

TAN Returns tangent.

radians A numeric constant, variable or expression for the angle in radians.

The **TAN** procedure returns the trigonometric tangent of an angle measured in *radians*. The tangent is the ratio of the angle's opposite side divided by its adjacent side.

Return Data Type: REAL

Example:

```
Angle = 45 * Deg2Rad           !Translate 45 degrees to Radians
TangentAngle = TAN(Angle)      !Get the tangent of 45 degree angle
```

See Also:

ATAN

SIN

ASIN

COS

ACOS

ASIN (return arcsine)

ASIN(*expression***)**

ASIN Returns inverse sine.

expression A numeric constant, variable, or expression for the value of the sine.

The **ASIN** procedure returns the inverse sine. The inverse of a sine is the angle that produces the sine. The return value is the angle in radians.

Return Data Type: REAL

Example:

```
InvSine = ASIN(SineAngle)     !Get the Arcsine
```

See Also:

TAN

ATAN

SIN

COS

ACOS

ACOS (return arccosine)

ACOS(*expression***)**

ACOS Returns inverse cosine.

expression A numeric constant, variable, or expression for the value of the cosine.

The **ACOS** procedure returns the inverse cosine. The inverse of a cosine is the angle that produces the cosine. The return value is the angle in radians.

Return Data Type: REAL

Example:

```
InvCosine = ACOS(CosineAngle)    !Get the Arccosine
```

See Also:

TAN

ATAN

SIN

ASIN

COS

ATAN (return arctangent)

ATAN(*expression***)**

ATAN Returns inverse tangent.

expression A numeric constant, variable, or expression for the value of the tangent.

The **ATAN** procedure returns the inverse tangent. The inverse of a tangent is the angle that produces the tangent. The return value is the angle in radians.

Return Data Type REAL

Example:

```
InvTangent = ATAN(TangentAngle)    !Get the Arctangent
```

See Also:

TAN

SIN

ASIN

COS

ACOS

String Procedures

ALL (return repeated characters)

ALL(*string* [,*length*]**)**

ALL	Returns repeated characters.
string	A string expression containing the character sequence to be repeated.
length	The length of the return string. If omitted the *length* of the return string is 255 characters.

The **ALL** procedure returns a string containing repetitions of the character sequence *string*.

Return Data Type: STRING

Example:

```
Starline  = ALL('*',25)       !Get 25 asterisks
Dotline   = ALL('.')          !Get 255 dots
```

CENTER (return centered string)

CENTER(*string* [,*length*]**)**

CENTER Returns centered string.

string A string constant, variable or expression.

length The length of the return string. If omitted, the length of the *string* parameter is used.

The **CENTER** procedure first removes leading and trailing spaces from a *string*, then pads it with leading and trailing spaces to center it within the *length*, and returns a centered string.

Return Data Type: STRING

Example:

```
!CENTER('ABC',5)     returns ' ABC '
!CENTER('ABC  ')     returns ' ABC '
!CENTER(' ABC')      returns ' ABC '

 Message = CENTER(Message)      !Center the message
Rpt:Title = CENTER(Name,60)     !Center the name
```

See Also:

LEFT

RIGHT

CHOOSE (return chosen value)

CHOOSE(| *expression* ,*value*, *value* [,*value...*] | **)**
| *condition* ,*value*, *value* |

CHOOSE Returns the chosen value from a list of possible values.

expression An arithmetic expression which determines which *value* parameter to return. This expression must resolve to a positive integer.

value A variable, constant, or expression for the procedure to return.

condition A logical expression which determines which of the two required *value* parameters to return. When the *expression* is true, the first *value* is returned, and when false, the second *value* is returned.

The **CHOOSE** procedure evaluates the *expression* or *condition* and returns the appropriate *value* parameter. If the *expression* resolves to a positive integer, that integer selects the corresponding *value* parameter for the CHOOSE procedure to return. If the *expression* evaluates to an out-of-range integer, then CHOOSE returns the last *value* parameter.

When the *condition* evaluates to true, then CHOOSE returns the first *value* parameter. When the *condition* evaluates to false, then CHOOSE returns the second *value* parameter.

The return data type is dependent upon the data types of the *value* parameters:

All Value Parameters	Return Data Type
LONG	LONG
DECIMAL or LONG	DECIMAL
STRING	STRING
DECIMAL, LONG, or STRING	DECIMAL
anything else	REAL

Return Data Type: LONG, DECIMAL, STRING, or REAL

Example:

```
!CHOOSE(4,'A','B','C','D','E') returns 'D'
!CHOOSE(1 > 2,'A','B') returns 'B'

?MyControl{-PROP:Hide} = CHOOSE(SomeField = 0,TRUE,FALSE)
                      !Hide or unhide control, based on the value in SomeField

MyView{-PROP:Filter}   = 'Weight > CHOOSE(Sex = ''M'',180,120)'
                      !VIEW filter to select "overweight" people of both sexes
```

See Also:

INLIST

CHR (return character from ASCII)

CHR(*code***)**

CHR Returns the display character.

code A numeric expression containing a numeric ASCII character code.

The **CHR** procedure returns the ANSI character represented by the ASCII character *code* parameter.

Return Data Type: STRING

Example:

```
Stringvar = CHR(122)        !Get lower case z
Stringvar = CHR(65)         !Get upper case A
```

See Also:

VAL

CLIP (return string without trailing spaces)

CLIP(*string***)**

CLIP Removes trailing spaces.

string A string expression.

The **CLIP** procedure removes trailing spaces from a *string*. The return string is a substring with no trailing spaces. CLIP is frequently used with the concatenation operator in string expressions using STRING data types.

CLIP is not normally needed with CSTRING data types, since these have a terminating character. CLIP is also not normally needed with PSTRING data types, since these have a length byte.

When used in conjunction with the LEFT procedure, you can remove both leading and trailing spaces (frequently called ALLTRIM in other languages).

Return Data Type: STRING

Example:

```
Name = CLIP(Last) & ', ' & CLIP(First) & Init & '.' !Full name in military order

AllTrimVar = CLIP(LEFT(MyVar))     !Trim leading and trailing spaces at once
```

See Also:

LEFT

DEFORMAT (return unformatted numbers from string)

DEFORMAT(*string* [,*picture*]**)**

DEFORMAT	Removes formatting characters from a numeric string.
string	A string expression containing a numeric string.
picture	A picture token, or the label of a CSTRING variable containing a picture token. If omitted, the picture for the *string* parameter is used. If the *string* parameter was not declared with a picture token, the return value will contain only characters that are valid for a numeric constant.

The **DEFORMAT** procedure removes formatting characters from a numeric string, returning only the numbers contained in the string. When used with a date or time *picture* (except those containing alphabetic characters), it returns a STRING containing the Clarion Standard Date or Time.

Return Data Type: STRING

Example:

```
  !DEFORMAT('$1,234.56')          returns 1234.56
  !DEFORMAT('309-53-9954')        returns 309539954
  !DEFORMAT('40A1-7',@P##A1-#P)   returns 407

DialString = 'ATDT1' & DEFORMAT(Phone,@P(###)###-####P) & '<13,10>'
                                  !Get phone number for modem to dial
ClarionDate = DEFORMAT(dBaseDate,@D1)  !Clarion Standard date from mm/dd/yy string

Data = '45,123'                        !Assign a formatted number to a string
Number = DEFORMAT(Data)                ! then remove non-numeric characters
```

See Also:

FORMAT

Standard Date

Standard Time

FORMAT (return formatted numbers into a picture)

FORMAT(*value*,*picture***)**

FORMAT Returns a formatted numeric string.

value A numeric expression for the *value* to be formatted.

picture A picture token or the label of a CSTRING variable containing a picture token.

The **FORMAT** procedure returns a numeric string formatted according to the *picture* parameter.

Return Data Type: STRING

Example:

```
Rpt:SocSecNbr = FORMAT(Emp:SSN,@P###-##-####P)             !Format the soc-sec-nbr
Phone = FORMAT(DEFORMAT(Phone,@P###-###-####P),@P(###)###-####P)
                                 !Change phone format from dashes to parens
DateString = FORMAT(DateLong,@D1)     !Format a date as a string
```

See Also:

DEFORMAT

INLIST (return entry in list)

INLIST(*searchstring*,*liststring*,*liststring* [,*liststring...*]**)**

INLIST Returns item in a list.

searchstring A constant, variable, or expression that contains the value for which to search. If the value is numeric, it is converted to a string before comparisons are made.

liststring The label of a variable or constant value to compare against the *searchstring*. If the value is numeric, it is converted to a string before comparisons are made. There may be up to 16 *liststring* parameters, and there must be at least two.

The **INLIST** procedure compares the contents of the *searchstring* against the values contained in each *liststring* parameter. If a matching value is found, the procedure returns the number of the first *liststring* parameter containing the matching value (relative to the first *liststring* parameter). If the *searchstring* is not found in any *liststring* parameter, **INLIST** returns zero.

Return Data Type: LONG

Example:

```
!INLIST('D','A','B','C','D','E') returns 4
 !INLIST('B','A','B','C','D','E') returns 2

 EXECUTE INLIST(Emp:Status,'Fulltime','Parttime','Retired','Consultant')
   Scr:Message = 'All Benefits'           !Full timer
   Scr:Message = 'Holidays Only'          !Part timer
   Scr:Message = 'Medical/Dental Only'     !Retired
   Scr:Message = 'No Benefits'            !Consultant
 END
```

See Also:

CHOOSE

INSTRING (return substring position)

INSTRING(*substring*,*string* [,*step*] [,*start*]**)**

INSTRING Searches for a substring in a string.

substring A string constant, variable, or expression that contains the string for which to search. You should CLIP a variable *substring* so INSTRING will not look for a match that contains the trailing spaces in the variable.

string A string constant, or the label of the STRING, CSTRING, or PSTRING variable to be searched.

step A numeric constant, variable, or expression which specifies the step length of the search. A *step* of 1 searches for the *substring* beginning at every character in the *string*, a *step* of 2 starts at every other character, and so on. If *step* is omitted, the step length defaults to the length of the *substring*.

start A numeric constant, variable, or expression which specifies where to begin the search of the *string*. If omitted, the search starts at the first character position.

The **INSTRING** procedure *steps* through a *string*, searching for the occurrence of a *substring*. If the *substring* is found, the procedure returns the *step* number on which the *substring* was found. If the *substring* is not found in the *string*, **INSTRING** returns zero.

Return Data Type: UNSIGNED

Example:

```
!INSTRING('DEF','ABCDEFGHIJ',1,1) returns 4
!INSTRING('DEF','ABCDEFGHIJ',2,1) returns 0
!INSTRING('DEF','ABCDEFGHIJ',2,2) returns 2
!INSTRING('DEF','ABCDEFGHIJ',3,1) returns 2

 Extension = SUB(FileSpec,INSTRING('.',FileSpec) + 1,3)
                                   !Extract extension from file spec

 IF INSTRING(CLIP(Search),Cus:Notes,1,1)   !If search variable found
   Scr:Message = 'Found'                   ! display message
 END
```

See Also:

SUB

STRING

CSTRING

PSTRING

String Slicing

LEFT (return left justified string)

LEFT(*string* [,*length*]**)**

LEFT	Left justifies a string.
string	A string constant, variable, or expression.
length	A numeric constant, variable, or expression for the length of the return string. If omitted, *length* defaults to the length of the *string*.

The **LEFT** procedure returns a left justified string. Leading spaces are removed from the *string*.

Return Data Type: STRING

Example:

```
 !LEFT('  ABC') returns 'ABC  '

CompanyName = LEFT(CompanyName)    !Left justify the company name
```

See Also:

RIGHT

CENTER

LEN (return length of string)

LEN(*string***)**

LEN Returns length of a string.

string A string constant, variable, or expression.

The **LEN** procedure returns the length of a *string*. If the *string* parameter is the label of a STRING variable, the procedure will return the declared length of the variable. If the *string* parameter is the label of a CSTRING or PSTRING variable, the procedure will return the length of the contents of the variable. Numeric variables are automatically converted to STRING intermediate values.

Return Data Type: UNSIGNED

Example:

```
IF LEN(CLIP(Title) & ' ' & CLIP(First) & ' ' & CLIP(Last)) > 30
                                             !If full name won't fit
  Rpt:Name = CLIP(Title) & ' ' & SUB(First,1,1) & '. ' & Last
                                             ! use first initial
ELSE
  Rpt:Name = CLIP(Title) & ' ' & CLIP(First) & ' ' & CLIP(Last)
                                             ! else use full name
END
Rpt:Title = CENTER(Cus:Name,LEN(Rpt:Title))         !Center the name in the title
```

LOWER (return lower case)

LOWER(*string***)**

LOWER Converts a string to all lower case.

string A string constant, variable, or expression for the *string* to be converted.

The **LOWER** procedure returns a string with all letters converted to lower case.

Return Data Type: STRING

Example:

```
!LOWER('ABC') returns 'abc'
Name = SUB(Name,1,1) & LOWER(SUB(Name,2,19)) !Make the rest of the name lower case
```

See Also:

UPPER

ISUPPER

ISLOWER

NUMERIC (return numeric string)

NUMERIC(*string***)**

NUMERIC Validates all numeric string.

string A string constant, variable, or expression.

The **NUMERIC** procedure returns the value 1 (true) if the *string* only contains a valid numeric value. It returns zero (false) if the *string* contains any non-numeric characters. Valid numeric characters are the digits 0 through 9, a leading minus sign, and a decimal point. DEFORMAT is used to return unformatted numbers from a formatted string.

Return Data Type: UNSIGNED

Example:

```
 !NUMERIC('1234.56')   returns 1
 !NUMERIC('1,234.56')  returns 0
 !NUMERIC('-1234.56')  returns 1
 !NUMERIC('1234.56-')  returns 0

IF NOT NUMERIC(PartNumber)      !If part number is not numeric
 DO ChkValidPart                ! check for valid part number
END                             !End if
```

See Also:

DEFORMAT

RIGHT (return right justified string)

RIGHT(*string* [,*length*]**)**

RIGHT Right justifies a string.

string A string constant, variable, or expression.

length A numeric constant, variable, or expression for the length of the return string. If omitted, the *length* is set to the length of the *string*.

The **RIGHT** procedure returns a right justified string. Trailing spaces are removed, then the string is right justified and returned with leading spaces.

Return Data Type: STRING

Example:

```
  !RIGHT('ABC  ') returns '  ABC'

Message = RIGHT(Message)      !Right justify the message
```

See Also:

LEFT

CENTER

SUB (return substring of string)

SUB(*string*,*position*,*length***)**

SUB Returns a portion of a string.

string A string constant, variable or expression.

position A integer constant, variable, or expression. If positive, it points to a character position relative to the beginning of the *string*. If negative, it points to the character position relative to the end of the *string* (i.e., a *position* value of -3 points to a position 3 characters from the end of the *string*).

length A numeric constant, variable, or expression of number of characters to return.

The **SUB** procedure parses out a sub-string from a *string* by returning *length* characters from the *string*, starting at *position*.

The SUB procedure is similar to the "string slicing" operation on STRING, CSTRING, and PSTRING variables. SUB is less flexible and efficient than string slicing, but SUB is "safer" because it ensures that the operation does not overflow the bounds of the *string*.

"String slicing" is more flexible than SUB because it may be used on both the destination and source sides of an assignment statement, while the SUB procedure can only be used as the source. It is more efficient because it takes less memory than individual character assignments or the SUB procedure (however, no bounds checking occurs).

To take a "slice" of a string, the beginning and ending character numbers are separated by a colon (:) and placed in the implicit array dimension position within the square brackets ([]) of the string. The position numbers may be integer constants, variables, or expressions. If variables are used, there must be at least one blank space between the variable name and the colon separating the beginning and ending number (to prevent PREfix confusion).

Return Data Type: STRING

Example:

```
 !SUB('ABCDEFGHI',1,1) returns 'A'
 !SUB('ABCDEFGHI',-1,1)returns 'I'
 !SUB('ABCDEFGHI',4,3) returns 'DEF'
  Extension = SUB(FileName,INSTRING('.',FileName,1,1)+1,3)
                         !Get the file extension using SUB procedure
  Extension = FileName[(INSTRING('.',FileName,1,1)+1) :
(INSTRING('.',FileName,1,1)+3)]
                         !The same operation using string slicing
```

See Also:

INSTRING

STRING

CSTRING

PSTRING

String Slicing

UPPER (return upper case)

UPPER(*string***)**

UPPER Returns all upper case string.

string A string constant, variable, or expression for the *string* to be converted.

The **UPPER** procedure returns a string with all letters converted to upper case.

Return Data Type: STRING

Example:

```
!UPPER('abc') returns 'ABC'
Name = UPPER(Name)    !Make the name upper case
```

See Also:

LOWER

ISUPPER

ISLOWER

VAL (return ASCII value)

VAL(*character***)**

VAL Returns ASCII code.

character A one-byte string containing an ANSI character.

The **VAL** procedure returns the ASCII code of a *character*.

Return Data Type: LONG

Example:

```
 !VAL('A')    returns 65
 !VAL('z')    returns 122

CharVal = VAL(StrChar)        !Get the ASCII value of the string character
```

See Also:

CHR

Bit Manipulation Procedures

BAND (return bitwise AND)

BAND(*value*,*mask***)**

BAND	Performs bitwise AND operation.
value	A numeric constant, variable, or expression for the bit *value* to be compared to the bit *mask*. The *value* is converted to a LONG data type prior to the operation, if necessary.
mask	A numeric constant, variable, or expression for the bit *mask*. The *mask* is converted to a LONG data type prior to the operation, if necessary.

The **BAND** procedure compares the *value* to the *mask*, performing a Boolean AND operation on each bit. The return value is a LONG integer with a one (1) in the bit positions where the *value* and the *mask* both contain one (1), and zeroes in all other bit positions.

BAND is usually used to determine whether an individual bit, or multiple bits, are on (1) or off (0) within a variable.

Return Data Type: LONG

Example:

```
!BAND(0110b,0010b) returns 0010b !0110b = 6, 0010b = 2

RateType  BYTE                   !Type of rate
Female  EQUATE(0001b)            !Female mask
Male      EQUATE(0010b)          !Male mask
Over25  EQUATE(0100b)            !Over age 25 mask
  CODE
  IF BAND(RateType,Female)   |      !If female
     AND BAND(RateType,Over25)      ! and over 25
    DO BaseRate                     ! use base premium
  ELSIF BAND(RateType,Male)         !If male
    DO AdjBase                      ! adjust base premium
  END
```

See Also:

BOR

BXOR

BSHIFT

BOR (return bitwise OR)

BOR(*value*,*mask***)**

BOR Performs bitwise OR operation.

value A numeric constant, variable, or expression for the bit *value* to be compared to the bit *mask*. The *value* is converted to a LONG data type prior to the operation, if necessary.

mask A numeric constant, variable, or expression for the bit *mask*. The *mask* is converted to a LONG data type prior to the operation, if necessary.

The **BOR** procedure compares the *value* to the *mask*, performing a Boolean OR operation on each bit. The return value is a LONG integer with a one (1) in the bit positions where the *value*, or the *mask*, or both, contain a one (1), and zeroes in all other bit positions.

BOR is usually used to unconditionally turn on (set to one), an individual bit, or multiple bits, within a variable.

Return Data Type: LONG

Example:

```
!BOR(0110b,0010b) returns 0110b  !0110b = 6, 0010b = 2

RateType  BYTE                    !Type of rate
Female   EQUATE(0001b)            !Female mask
Male     EQUATE(0010b)            !Male mask
Over25   EQUATE(0100b)            !Over age 25 mask
  CODE
  RateType = BOR(RateType,Over25)   !Turn on over 25 bit
  RateType = BOR(RateType,Male)     !Set rate to male
```

See Also:

BAND

BXOR

BSHIFT

BXOR (return bitwise exclusive OR)

BXOR(*value*,*mask***)**

BXOR	Performs bitwise exclusive OR operation.
value	A numeric constant, variable, or expression for the bit *value* to be compared to the bit *mask*. The *value* is converted to a LONG data type prior to the operation, if necessary.
mask	A numeric constant, variable, or expression for the bit *mask*. The *mask* is converted to a LONG data type prior to the operation, if necessary.

The **BXOR** procedure compares the *value* to the *mask*, performing a Boolean XOR operation on each bit. The return value is a LONG integer with a one (1) in the bit positions where either the *value* or the *mask* contain a one (1), but not both. Zeroes are returned in all bit positions where the bits in the *value* and *mask* are alike.

BXOR is usually used to toggle on (1) or off (0) an individual bit, or multiple bits, within a variable.

Return Data Type: LONG

Example:

```
!BXOR(0110b,0010b) returns 0100b  !0110b = 6, 0100b = 4, 0010b = 2

RateType  BYTE                   !Type of rate
Female   EQUATE(0001b)           !Female mask
Male     EQUATE(0010b)           !Male mask
Over25   EQUATE(0100b)           !Over age 25 mask
Over65   EQUATE(1100b)           !Over age 65 mask
  CODE
  RateType = BXOR(RateType,Over65)   !Toggle over 65 bits
```

See Also:

BAND

BOR

BSHIFT

BSHIFT (return shifted bits)

BSHIFT(*value*,*count***)**

BSHIFT	Performs the bit shift operation.
value	A numeric constant, variable, or expression. The *value* is converted to a LONG data type prior to the operation, if necessary.
count	A numeric constant, variable, or expression for the number of bit positions to be shifted. If *count* is positive, *value* is shifted left. If *count* is negative, *value* is shifted right.

The **BSHIFT** procedure shifts a bit *value* by a bit *count*. The bit value may be shifted left (toward the high order), or right (toward the low order). Zero bits are supplied to fill vacated bit positions when shifting.

Return Data Type: LONG

Example:

```
  !BSHIFT(0110b,1)    returns 1100b
  !BSHIFT(0110b,-1)   returns 0011b

Varswitch = BSHIFT(20,3)            !Multiply by eight
Varswitch = BSHIFT(Varswitch,-2)    !Divide by four
```

See Also:

BAND

BOR

BXOR

Date / Time Procedures

Standard Date

A Clarion standard date is the number of days that have elapsed since December 28, 1800. The range of accessible dates is from January 1, 1801 (standard date 4) to December 31, 9999 (standard date 2,994,626). Date procedures will not return correct values outside the limits of this range. The standard date calendar also adjusts for each leap year within the range of accessible dates. Dividing a standard date by modulo 7 gives you the day of the week: zero = Sunday, one = Monday, etc.

The LONG data type with a date format (@D) display picture is normally used for a standard date. Data entry into any date format picture with a two-digit year defaults to the century of next 20 or previous 80 years. For example, entering 01/01/01 results in 01/01/2001 if the current year (per the system clock) is greater than 1980, and 01/01/1901 if the current year is 1980 or earlier.

The DATE data type is a data format used in the Btrieve Record Manager and some other file systems. A DATE field is internally converted to LONG containing the Clarion standard date before any mathematical or date procedure operation is performed. Therefore, DATE should be used for external file compatibility, and LONG is normally used for other dates.

See Also:

DAY

MONTH

YEAR

TODAY

DATE

Standard Time

A Clarion standard time is the number of hundredths of a second that have elapsed since midnight, plus one (1). The valid range is from 1 (defined as midnight) to 8,640,000 (defined as 11:59:59.99 PM). A standard time of one is exactly equal to midnight to allow a zero value to be used to detect no time entered. Although time is expressed to the nearest hundredth of a second, the system clock is only updated 18.2 times a second (approximately every 5.5 hundredths of a second).

The LONG data type with a time format (@T) display picture is normally used for a standard time. The TIME data type is a data format used in the Btrieve Record Manager. A TIME field is internally converted to LONG containing the Clarion standard time before any mathematical or time procedure operation is performed. Therefore, TIME should be used for external Btrieve file compatibility, and LONG should normally be used for other times.

See Also:

CLOCK

TODAY (return system date)

TODAY( )

The **TODAY** procedure returns the operating system date as a standard date. The range of possible dates is from January 1, 1801 (standard date 4) to December 31, 2099 (standard date 109,211).

Return Data Type: LONG

Example:

```
OrderDate = TODAY()     !Set the order date to system date
```

See Also:

Standard Date

DAY

MONTH

YEAR

SETTODAY

DATE

CLOCK (return system time)

CLOCK( )

The **CLOCK** procedure returns the time of day from the operating system time in standard time (expressed as hundredths of a second since midnight, plus one). Although the time is expressed to the nearest hundredth of a second, the system clock is only updated 18.2 times a second (approximately every 5.5 hundredths of a second).

Return Data Type: LONG

Example:

```
Time = CLOCK()      !Save the system time
```

See Also:

Standard Time

SETCLOCK

DATE (return standard date)

DATE(*month*,*day*,*year***)**

DATE	Return standard date.
month	A positive numeric constant, variable, or expression for the *month*.
day	A positive numeric constant, variable, or expression for the *day* of the month.
year	A numeric constant, variable or expression for the *year*. The valid range for a *year* value is 00 through 99 (using "Intellidate" logic), or 1801 through 2099.

The **DATE** procedure returns a standard date for a given *month*, *day*, and *year*. The *month* and *day* parameters do allow positive out-of-range values (zero or negative values are invalid). A *month* value of 13 is interpreted as January of the next year. A *day* value of 32 in January is interpreted as the first of February. Consequently, DATE(12,32,97), DATE(13,1,97), and DATE(1,1,98) all produce the same result.

The century for a two-digit *year* parameter is resolved using the default "Intellidate" logic, which assumes the date falls in the range of the next 20 or previous 80 years from the current operating system date. For example, assuming the current year is 1998, if the *year* parameter is "15," the date returned is in the year 2015, and if the *year* parameter is "60," the date returned is in 1960.

Return Data Type: LONG

Example:

```
HireDate = DATE(Hir:Month,Hir:Day,Hir:Year)           !Compute hire date
FirstOfMonth = DATE(MONTH(TODAY()),1,YEAR(TODAY()))   !Compute First day of month
```

See Also:

Standard Date

DAY

MONTH

YEAR

TODAY

DAY (return day of month)

DAY(*date***)**

DAY Returns day of month.

date A numeric constant, variable, expression, or the label of a STRING, CSTRING, or PSTRING variable declared with a date picture token. The *date* must be a standard date. A variable declared with a date picture token is automatically converted to a standard date intermediate value.

The **DAY** procedure computes the day of the month (1 to 31) for a given standard date.

Return Data Type: LONG

Example:

```
OutDay = DAY(TODAY())    !Get the day from today's date
DueDay = DAY(TODAY()+2)  !Calculate the return day
```

See Also:

Standard Date

MONTH

YEAR

TODAY

DATE

MONTH (return month of date)

MONTH(*date***)**

MONTH Returns month in year.

date A numeric constant, variable, expression, or the label of a STRING, CSTRING, or PSTRING variable declared with a date picture token. The *date* must be a standard date. A variable declared with a date picture token is automatically converted to a standard date intermediate value.

The **MONTH** procedure returns the month of the year (1 to 12) for a given standard date.

Return Data Type: LONG

Example:

```
PayMonth = MONTH(DueDate)     !Get the month from the date
```

See Also:

Standard Date

DAY

YEAR

TODAY

DATE

YEAR (return year of date)

YEAR(*date***)**

YEAR Returns the year.

date A numeric constant, variable, expression, or the label of a string variable declared with a date picture, containing a standard date. A variable declared with a date picture is automatically converted to a standard date intermediate value.

The **YEAR** procedure returns a four digit number for the year of a standard *date* (1801 to 9999).

Return Data Type: LONG

Example:

```
IF YEAR(LastOrd) < YEAR(TODAY())    !If last order date not from this year
  DO StartNewYear                   ! start new year to date totals
END
```

See Also:

Standard Date

DAY

MONTH

TODAY

DATE

AGE (return age from base date)

AGE(*birthdate* [,*base date*]**)**

AGE Returns elapsed time.

birthdate A numeric expression for a standard date.

base date A numeric expression for a standard date. If this parameter is omitted, the operating system date is used for the computation.

The **AGE** procedure returns a string containing the time elapsed between two dates. The age return string is in the following format:

```
1 to 60 days          - 'nn DAYS'
61 days to 24 months - 'nn MOS'
2 years to 999 years - 'nnn YRS'
```

Return Data Type: STRING

Example:

```
Message = Emp:Name & 'is ' & AGE(Emp:DOB,TODAY()) & ' old today.'
```

See Also:

Standard Date

DAY

MONTH

YEAR

TODAY

DATE

Picture Tokens

Picture tokens provide a masking format for displaying and editing variables. There are seven types of picture tokens: numeric and currency, scientific notation, string, date, time, pattern, and key-in template.

Numeric and Currency Pictures
Scientific Notation Pictures
String Pictures
Date Pictures
Time Pictures

Numeric and Currency Pictures

@N [*currency*] [*sign*] [*fill*] *size* [*grouping*] [*places*] [*sign*] [*currency*] [**B**]

@N All numeric and currency pictures begin with @N.

currency Either a dollar sign ($) or any string constant enclosed in tildes (~). When it precedes the *sign* indicator and there is no *fill* indicator, the *currency* symbol "floats" to the left of the high order digit. If there is a *fill* indicator, the *currency* symbol remains fixed in the left-most position. If the *currency* indicator follows the *size* and *grouping*, it appears at the end of the number displayed.

sign Specifies the display format for negative numbers. If a hyphen precedes the *fill* and *size* indicators, negative numbers will display with a leading minus sign. If a hyphen follows the *size*, *grouping*, *places*, and *currency* indicators, negative numbers will display with a trailing minus sign. If parentheses are placed in both positions, negative numbers will be displayed enclosed in parentheses. To prevent ambiguity, a trailing minus *sign* should always have *grouping* specified.

fill Specifies leading zeros, spaces, or asterisks (*) in any leading zero positions, and suppressesdefault *grouping*. If the *fill* is omitted, leading zeros are suppressed.

0 (zero) Produces leading zeroes
_ (underscore) Produces leading spaces
***** (asterisk) Produces leading asterisks

size The *size* is required to specify the total number of significant digits to display, including the number of digits in the *places* indicator and any formatting characters.

grouping A *grouping* symbol, other than a comma (the default), can appear right of the *size* indicator to specify a three digit group separator. To prevent ambiguity, a hyphen *grouping* indicator should also specify the *sign*.

. (period) Produces periods
- (hyphen) Produces hyphens
_ (underscore) Produces spaces

places Specifies the decimal separator symbol and the number of decimal digits. The number of decimal digits must be less than the *size*. The decimal separator may be a period (.), grave accent (') (produces periods *grouping* unless overridden), or the letter "v" (used only for STRING field storage declarationsnot for display).

. (period) Produces a period
' (grave accent) Produces a comma
v Produces no decimal separator

B Specifies blank display whenever its value is zero.

The numeric and currency pictures format numeric values for screen display or in reports. If the value is greater than the maximum value the picture can display, a string of pound signs (#) is displayed.

Example:

```
Numeric              Result   Format
@N9               4,550,000   Nine digits, group with commas (default)
@N_9B               4550000   Nine digits, no grouping, leading blanks if zero
@N09              004550000   Nine digits, leading zero
@N*9              ***45,000   Nine digits, asterisk fill, group with commas
@N9_              4 550 000   Nine digits, group with spaces
@N9.              4.550.000   Nine digits, group with periods

Decimal              Result   Format
@N9.2              4,550.75   Two decimal places, period decimal separator
@N_9.2B             4550.75   Two decimal places, period decimal separator, no
                              grouping, blank if zero
```

```
@N_9'2              4550,75   Two decimal places, comma decimal separator
@N9.'2             4.550,75   Comma decimal separator, group with periods
@N9_'2             4 550,75   Comma decimal separator, group with spaces,

Signed               Result   Format
@N-9.2B           -2,347.25   Leading minus sign, blank if zero
@N9.2-            2,347.25-   Trailing minus sign
@N(10.2)         (2,347.25)   Enclosed in parens when negative

Dollar Currency      Result   Format
@N$9.2B           $2,347.25   Leading dollar sign, blank if zero
@N$10.2-         $2,347.25-   Leading dollar sign, trailing minus when negative
@N$(11.2)       $(2,347.25)   Leading dollar sign, in parens when negative

Int'l Currency             Result      Format
@N12_'2~ F~           1 5430,50 F      France
@N~L. ~12'           L. 1.430.050      Italy
@N~£~12.2               £1,240.50      United Kingdom
@N~kr~12'2             kr1.430,50      Norway
@N~DM~12'2             DM1.430,50      Germany
@N12_'2~ mk~         1 430,50 mk       Finland
@N12'2~ kr~          1.430,50 kr       Sweden

Storage-Only Pictures:
Variable1 STRING(@N_6v2)           !Declare as 6 bytes stored without decimal
  CODE
  Variable1 = 1234.56              !Assign value, stores '123456' in file
  MESSAGE(FORMAT(Variable1,@N_7.2))!Display with decimal point: '1234.56'
```

Scientific Notation Pictures

@Emsn[**B**]

@E All scientific notation pictures begin with @E.

m Determines the total number of characters in the format provided by the picture.

s Specifies the decimal separation character, and the grouping character when the **n** value is greater than 3.

. (period)	period and comma
.. (period period)	period and period
' (grave accent)	comma and period
_.(underscore period)	period and space

n Indicates the number of digits that appear to the left of the decimal point.

B Specifies that the format displays as blank when the value is zero.

The scientific notation picture formats very large or very small numbers. The format is a decimal number raised by a power of ten.

Example:

Picture	Value	Result
@E9.0	1,967,865	.20e+007
@E12.1	1,967,865	1.9679e+006
@E12.1B	0	
@E12.1	-1,967,865	-1.9679e+006
@E12.1	.000000032	3.2000e-008
@E12_.4	1,967,865	1 967.865e+003

String Pictures

@**S**_length_

@S All string pictures begin with @S.

length Determines the number of characters in the picture format.

A string picture describes an unformatted string of a specific _length_.

Example:

```
Name STRING(@S20)   !A 20 character string field
```

Date Pictures

@Dn [**s**] [*direction* [*range*]] [**B**]

@D All date pictures begin with @D.

n Determines the date picture format. Date picture formats range from 1 through 18. A leading zero (0) indicates a zero-filled day or month.

s A separation character between the month, day, and year components. If omitted, the slash (/) appears.

. (period) Produces periods
' (grave accent) Produces commas
- (hyphen) Produces hyphens
_ (underscore) Produces spaces

direction A right or left angle bracket (> or <) that specifies the "Intellidate" direction (> indicates future, < indicates past) for the *range* parameter. Valid only on date pictures with two-digit years.

range An integer constant in the range of zero (0) to ninety-nine (99) that specifies the "Intellidate" century for the *direction* parameter. Valid only on date pictures with two-digit years. If omitted, the default value is 80.

B Specifies that the format displays as blank when the value is zero.

Dates may be stored in numeric variables (usually LONG), a DATE field (for Btrieve compatibility), or in a STRING declared with a date picture. A date stored in a numeric variable is called a "Clarion Standard Date." The stored value is the number of days since December 28, 1800. The date picture token converts the value into one of the date formats.

The century for dates in any picture with a two-digit year is resolved using "Intellidate" logic. Date pictures that do not specify *direction* and *range* parameters assume the date falls in the range of the next 20 or previous 80 years. The *direction* and *range* parameters allow you to change this default. The *direction* parameter specifies whether the *range* specifies the future or past value. The opposite *direction* then receives the opposite value (100-*range)* so that any two-digit year results in the correct century.

For example, the picture @D1>60 specifies using the appropriate century for each year 60 years in the future and 40 years in the past. If the current year is 1996, when the user enters "5/01/40," the date is in the year 2040, and when the user enters "5/01/60," the date is in the year 1960.

For those date pictures which contain month names, the actual names are customizable in an Environment file (.ENV). See the Internationalization section for more information.

Example:

Picture	Format	Result
@D1	mm/dd/yy	10/31/59
@D1>40	mm/dd/yy	10/31/59
@D01	mm/dd/yy	01/01/95
@D2	mm/dd/yyyy	10/31/1959
@D3	mmm dd, yyyy	OCT 31,1959
@D4	mmmmmmmmm dd, yyyy	October 31, 1959
@D5	dd/mm/yy	31/10/59
@D6	dd/mm/yyyy	31/10/1959
@D7	dd mmm yy	31 OCT 59
@D8	dd mmm yyyy	31 OCT 1959
@D9	yy/mm/dd	59/10/31
@D10	yyyy/mm/dd	1959/10/31
@D11	yymmdd	591031
@D12	yyyymmdd	19591031
@D13	mm/yy	10/59
@D14	mm/yyyy	10/1959

```
@D15                 yy/mm            59/10
@D16               yyyy/mm            1959/10
@D17   Windows Control Panel setting for Short Date
@D18   Windows Control Panel setting for Long Date Alternate separators

@D1.   mm.dd.yy   Period separator
@D2-   mm-dd-yyyy Dash separator
@D5_   dd mm yy   Underscore produces space separator
@D6'   dd,mm,yyyy Grave accent produces comma separator
```

See Also:

Standard Date

FORMAT

DEFORMAT

Environment Files

Time Pictures

@Tn[**s**][**B**]

@T All time pictures begin with @T.

n Determines the time picture format. Time picture formats range from 1 through 8. A leading zero (0) indicates zero-filled hours.

s A separation character. By default, colon (:) characters appear between the hour, minute, and second components of certain time picture formats. The following s indicators provide an alternate separation character for these formats.

. (period) Produces periods
' (grave accent) Produces commas
- (hyphen) Produces hyphens
_ (underscore) Produces spaces

B Specifies that the format displays as blank when the value is zero.

Times may be stored in a numeric variable (usually a LONG), a TIME field (for Btrieve compatibility), or in a STRING declared with a time picture. A time stored in a numeric variable is called a "Standard Time." The stored value is the number of hundredths of a second since midnight. The picture token converts the value to one of the eight time formats.

For those time pictures which contain string data, the actual strings are customizable in an Environment file (.ENV). See the Internationalization section for more information.

Example:

```
Picture              Format    Result
@T1                   hh:mm     17:30
@T2                    hhmm      1730
@T3                 hh:mmXM    5:30PM
@T03                hh:mmXM   05:30PM
@T4                hh:mm:ss  17:30:00
@T5                  hhmmss    173000
@T6              hh:mm:ssXM 5:30:00PM
@T7    Windows Control Panel setting for Short Time
@T8    Windows Control Panel setting for Long Time

   Alternate separators
@T1.   hh.mm    Period separator
@T1-   hh-mm    Dash separator
@T3_   hh mmXM  Underscore produces space separator
@T4'   hh,mm,ss Grave accent produces comma separator
```

See Also:

Standard Time

FORMAT

DEFORMAT

Environment Files

Special Characters

Initiators:

- ! Exclamation point begins a source code comment.
- ? Question mark begins a field equate label.
- @ At sign begins a picture token.
- * Asterisk begins a parameter passed by address in a MAP prototype.
- ~ A leading tilde on a filename indicates a file linked into the project.

Terminators:

- ; Semi-colon is an executable statement separator.
- CR/LF Carriage-return/Line-feed is an executable statement separator.
- . Period terminates a data or code structure (a substitute for END).
- | Vertical bar is the source code line continuation character.
- # Pound sign declares an implicit LONG variable.
- $ Dollar sign declares an implicit REAL variable.
- Double quote declares an implicit STRING variable.

Delimiters:

- ( ) Parentheses enclose a parameter list.
- [ ] Brackets enclose an array subscript list.
- Single quotes enclose a string constant.
- {- } Curly braces enclose a repeat count in a string constant, or a property parameter.
- < > Angle brackets enclose an ASCII code in a string constant, or indicate a parameter in a MAP prototype which may be omitted.
- : Colon separates the start and stop positions of a string slice.
- , Comma separates parameters in a parameter list.

Connecters:

- . Period is a decimal point used in numeric constants, or connects a complex structure label to the label of one of its members.
- : Colon connects a prefix to a label.
- $ Dollar sign connects the WINDOW or REPORT label to a field equate label in a controls property assignment expression.

Operators:

- + Plus sign indicates addition.
- - Minus sign indicates subtraction.
- * Asterisk indicates multiplication.
- / Slash indicates division.
- % Percent sign indicates modulus division.
- ^ Carat indicates exponentiation.
- < Left angle bracket indicates less than.
- > Right angle bracket indicates greater than.
- = Equal sign indicates assignment or equivalence.
- ~ Tilde indicates the logical (Boolean) NOT operator.
- & Ampersand indicates concatenation.
- &= Ampersand equal indicates reference assignment or reference equivalence.

Expressions

An expression is a mathematical, string, or logical formula that produces a value. An expression may be the source variable of an assignment statement, a parameter of a procedure, a subscript of an array (a dimensioned variable), or the condition of an IF, CASE, LOOP, or EXECUTE structure. Expressions may contain constant values, variables, and procedures which return values, all connected by logical and/or arithmetic or string operators.

Expression Evaluation
Arithmetic Operators
Logical Operators
Numeric Constants
Numeric Expressions
String Constants
The Concatenation Operator
String Expressions
Implicit String Arrays and String Slicing
Logical Expressions

Expression Evaluation

Expressions are evaluated in the standard algebraic order of operations. The precedence of operations is controlled by operator type and placement of parentheses. Each operation produces an (internal) intermediate value used in subsequent operations. Parentheses may be used to group operations within expressions. Expressions are evaluated beginning with the inner-most set of parentheses and working through to the outer-most set.

Precedence levels for expression evaluation, from highest to lowest, and left-to-right within each level, are:

Level 1	( )	Parenthetical Grouping
Level 2	-	Unary Minus (Negative sign)
Level 3	procedure call	Gets the RETURN value
Level 4	^	Exponentiation
Level 5	* / %	Multiplication, Division, Modulus Division
Level 6	+ -	Addition, Subtraction
Level 7	&	Concatenation
Level 8	= <>	Logical Comparisons
Level 9	NOT, AND, OR/XOR	Boolean expressions

Expressions may produce numeric values, string values, or logical values (true/false evaluation). An expression may contain no operators at all; it may be a single variable, constant value, or procedure call which returns a value.

Arithmetic Operators

An arithmetic operator combines two operands arithmetically to produce an intermediate value. The operators are:

+	Addition (A **+** B gives the sum of A and B)
-	Subtraction (A **-** B gives the difference of A and B)
*****	Multiplication (A ***** B multiples A by B)
/	Division (A **/** B divides A by B)
^	Exponentiation (A **^** B raises A to power of B)
%	Modulus Division (A **%** B gives the remainder of A divided by B)

Logical Operators

A logical operator compares two operands or expressions and produces a true or false condition. There are two types of logical operators: conditional and Boolean. Conditional operators compare two values or expressions. Boolean operators connect string, numeric, or logical expressions together to determine true-false logic. Operators may be combined to create complex operators.

Conditional Operators	=	Equal sign
	<	Less than
	>	Greater than
Boolean Operators	**NOT**	Boolean (logical) NOT
	~	Tilde (logical NOT)
	AND	Boolean AND
	OR	Boolean OR
	XOR	Boolean eXclusive OR
Combined operators	<>	Not equal
	~=	Not equal
	NOT =	Not equal
	<=	Less than or equal to
	=<	Less than or equal to
	~>	Not greater than
	NOT >	Not greater than
	>=	Greater than or equal to
	=>	Greater than or equal to
	~<	Not less than
	NOT <	Not less than

During logical evaluation, any non-zero numeric value or non-blank string value indicates a true condition, and a null (blank) string or zero numeric value indicates a false condition.

Example:

```
Logical Expression               Result
  A = B                          True when A is equal to B
  A < B                          True when A is less than B
  A > B                          True when A is greater than B
  A <> B, A ~= B, A NOT = B      True when A is not equal to B
  A ~< B, A >= B, A NOT < B      True when A is not less than B
  A ~> B, A <= B, A NOT > B      True when A is not greater than B
  ~ A, NOT A                     True when A is null or zero
  A AND B                        True when A is true and B is true
  A OR B                         True when A is true, or B is true, or both are
                                 true
  A XOR B                        True when A is true or B is true, but not both.
```

Numeric Constants

Numeric constants are fixed numeric values. They may occur in data declarations, in expressions, and as parameters of procedures or attributes. A numeric constant may be represented in decimal (base 10the default), binary (base 2), octal (base 8), hexadecimal (base 16), or scientific notation formats. Formatting characters, such as dollar signs and commas, are not permitted in numeric constants; only leading plus or minus signs and the decimal point are allowed.

Decimal (base ten) numeric constants may contain an optional leading minus sign (hyphen character), an integer, and an optional decimal with a fractional component. Binary (base two) numeric constants may contain an optional leading minus sign, the digits 0 and 1, and a terminating B or b character. Octal (base eight) numeric constants contain an optional leading minus sign, the digits 0 through 7, and a terminating O or o character. Hexadecimal (base sixteen) numeric constants contain an optional leading minus sign, the digits 0 through 9, alphabet characters A through F (representing the numbers 10 through 15) and a terminating H or h character. If the left-most character is a letter A through F, a leading zero must be used.

Example:

```
-924            !Decimal constants
76.346
+76.346
1011b           !Binary constants
-1000110B
3403o           !Octal constants
-70413120
-1FFBh          !Hexadecimal constants
0CD1F74FH
```

Numeric Expressions

Numeric expressions may be used as parameters of procedures, the condition of IF, CASE, LOOP, or EXECUTE structures, or as the source portion of an assignment statement where the destination is a numeric variable. A numeric expression may contain arithmetic operators and the concatenation operator, but they may not contain logical operators. When used in a numeric expression, string constants and variables are converted to numeric intermediate values. If the concatenation operator is used, the intermediate value is converted to numeric after the concatenation occurs.

Example:

```
Count + 1          !Add 1 to Count
(1 - N * N) / R    !N times N subtracted from 1 then divided by R
305 & 7854555      !Concatenate area code with phone number
```

See Also:

Data Conversion Rules

String Constants

A string constant is a set of characters enclosed in single quotes (apostrophes). The maximum length of a string constant is 255 characters. Characters that cannot be entered from the keyboard may be inserted into a string constant by enclosing their ASCII character codes in angle brackets (<>). ASCII character codes may be represented in decimal, hexadecimal, binary, or octal numeric constant format.

In a string constant, a left angle bracket (<) initiates a scan for a right angle bracket. Therefore, to include a left angle bracket in a string constant requires two left angle brackets in succession. To include an apostrophe as part of the value inside a string constant requires two apostrophes in succession. Two apostrophes (''), with no characters (or just spaces) between them, represents a null, or blank, string. Consecutive occurrences of the same character within a string constant may be represented by *repeat count* notation. The number of times the character is to be repeated is placed within curly braces ({- }) immediately following the character to repeat. To include a left curly brace ({-) as part of the value inside a string constant requires two left curly braces ({-{-) in succession.

The ampersand (&) is always valid in a string constant. However, depending on the assignment's destination, it may be interpreted as an underscore for a hot letter (for example, a PROMPT control's display *text*). In this case, you double it up (&&) to end up with a single ampersand in the screen display.

Example:

```
'string constant'      !A string constant
'It''s a girl!'        !With embedded apostrophe
'<27,15>'              !Using decimal ASCII codes
'A << B'               !With embedded left angle, A < B
'*{-20}'               !Twenty asterisks, repeat-count notation
''                     !A null (blank) string
```

The Concatenation Operator

The ampersand (&) concatenation operator is used to append one string or string variable to another. The length of the resulting string is the sum of the lengths of the two values being concatenated. Numeric data types may be concatenated with strings or other numeric variables or constants. In many cases, the CLIP procedure should be used to remove any trailing spaces from a string being concatenated to another string.

Example:

```
CLIP(FirstName) & ' ' Initial & '. ' & LastName       !Concatenate full name
'TopSpeed Corporation' & ', Inc.'                      !Concatenate two constants
```

See Also:

CLIP

Numeric Expressions

Data Conversion Rules

FORMAT

String Expressions

String expressions may be used as parameters of procedures and attributes, or as the source portion of an assignment statement when the destination is a string variable. String expressions may contain a single string or numeric variable, or a complex combination of sub-expressions, procedures, and operations.

Example:

```
StringVar   STRING(30)
Name        STRING(10)
Weight      STRING(3)
Phone       LONG
  CODE
  StringVar = 'Address:' & Cus:Address    !Concatenate a constant and variable

  StringVar = 'Phone:' & ' 305-' & FORMAT(Phone,@P###-####P)
                                     !Concatenate constant valuess
                                     ! and FORMAT procedure's return value

  StringVar = Weight & 'lbs.'             !Concatenate a constant and variable
```

See Also:

[CLIP]

[The Concatentaion Operator]

[Data Conversion Rules]

[FORMAT]

Implicit String Arrays and String Slicing

In addition to their explicit declaration, all STRING, CSTRING and PSTRING variables have an implicit array declaration of one character strings, dimensioned by the length of the string. This is directly equivalent to declaring a second variable as:

```
StringVar     STRING(10)
StringArray   STRING(1),DIM(SIZE(StringVar)),OVER(StringVar)
```

This implicit array declaration allows each character in the string to be directly addressed as an array element, without the need of the second declaration. The PSTRING's length byte is addressed as element zero (0) of the array, as is the first byte of a BLOB (the only two cases in Clarion where an array has a zero element).

If the string also has a DIM attribute, this implicit array declaration is the last (optional) dimension of the array (to the right of the explicit dimensions). The MAXIMUM procedure does not operate on the implicit dimension, you should use SIZE instead.

You may also directly address multiple characters within a string using the "string slicing" technique. This technique performs a similar function to the SUB procedure, but is much more flexible and efficient. It is more flexible because a "string slice" may be used as either the *destination* or *source* sides of an assignment statement, while the SUB procedure can only be used as the source. It is more efficient because it takes less memory than either individual character assignments or the SUB procedure.

To take a "slice" of the string, the beginning and ending character numbers are separated by a colon (:) and placed in the implicit array dimension position within the square brackets ([]) of the string. The position numbers may be integer constants, variables, or expressions (internally computed as LONG base type). If variables are used, there must be at least one blank space between the variable name and the colon separating the beginning and ending number (to prevent PREfix confusion).

Example:

```
Name     STRING(15)
CONTACT   STRING(15),DIM(4)
  CODE
  Name = 'Tammi'             !Assign a value
  Name[5] = 'y'              ! then change fifth letter
  Name[6] = 's'              ! then add a letter
  Name[0] = '<6>'            ! and handle length byte
  Name[5:6] = 'ie'           ! and change a "slice"  -- the fifth and sixth letters
  Contact[1] = 'First'       !Assign value to first element
  Contact[1,2] = 'u'         !Change first element 2nd character
  Contact[1,2:3] = Name[5:6] !Assign slice to first element 2nd & 3rd characters
```

Logical Expressions

Logical expressions evaluate true-false conditions in IF, LOOP UNTIL, and LOOP WHILE control structures. Control is determined by the final result (true or false) of the expression. Logical expressions are evaluated from left to right. The right operand of an AND, OR, or XOR logical expression will only be evaluated if it could affect the result. Parentheses should be used to eliminate ambiguous evaluation and to control evaluation precedence. The level or precedence for the logical operators is as follows:

Level 1	Conditional operators
Level 2	~, NOT
Level 3	AND
Level 4	OR, XOR

Example:

```
LOOP UNTIL KEYBOARD()       !True when user presses any key
 !some statements
END

IF A = B THEN RETURN.       !RETURN if A is equal to B

LOOP WHILE ~ Done#          !Loop while false (Done# = 0)
 !some statements
END

IF A >= B OR (C > B AND E = D) THEN RETURN.
                         !True if a >= b, also true if
                         ! both c > b and e = d.
                         !The second part of the expression
                         ! (after OR) is evaluated only if the
                         ! first part is not true.
```

Simple Assignment Statements

destination **=** *source*

destination The label of a variable or data structure property.

source A numeric or string constant, variable, procedure, expression, or data structure property.

The **=** sign assigns the value of *source* to the *destination*; it copies the value of the *source* expression into the *destination* variable. If *destination* and *source* are different data types, the value the *destination* receives from the *source* is dependent upon the Data Conversion Rules.

Example:

```
Name = 'JONES'                          !Variable = string constant
PI = 3.14159                            !Variable = numeric constant
Cosine = SQRT(1 - Sine * Sine)             !Variable = procedure return value
A = B + C + 3                         !Variable = numeric expression
Name = CLIP(FirstName) & ' ' Initial & '. ' & LastName
                                !Variable = string expression
```

See Also:

Data Conversion Rules

Operating Assignment Statements

destination	+=	*source*
destination	-=	*source*
destination	*=	*source*
destination	/=	*source*
destination	^=	*source*
destination	%=	*source*

destination Must be the label of a variable. This may not be any type property (window, control, report, etc.).

source A constant, variable, procedure, or expression.

Operating assignment statements perform their operation on the *destination* and *source,* assigning the result to the *destination*. Operating assignment statements are more efficient than their equivalent operations.

Example:

```
Operating AssignmentFunctional Equivalent
A += 1            A = A + 1
A -= B            A = A - B
A *= -5           A = A * -5
A /= 100          A = A / 100
A ^= I + 1        A = A ^ (I + 1)
A %= 7            A = A % 7
```

BCD Operations and Procedures

Clarion has a Binary Coded Decimal (BCD) library of operations and procedures that execute in a manner similar to the manner in which decimal arithmetic is performed on paper. These operations use internal intermediate values with 31 digits accuracy on both sides of the decimal point.

The big advantage of the BCD operations is that it is very easy to "see" what is happening because they execute just as you would with pencil and paper. Simply imagine doing the computation long hand and throwing away numbers that go off the end of the page (rounding to the right).

Having 31 fixed decimal places either side of the decimal point there are numbers that cannot be represented in a BCD system which can be represented by a REAL. Therefore, understanding what is going on is useful.

Generally, the only cases where underflow will affect you is in division operations, usually when dividing by a multiple of 3. For example:

```
100000/3 = 33333.3333333333333333333333333333
(100000/3)-INT(100000/3)*100000 =
33333.3333333333333333333333300000
```

BCD computation times are very data sensitive; the time taken is proportional to how long the computation would take you by hand. Therefore, the longer the numbers involved, the longer the execution times. However, standard "tricks of the trade" (such as multiplying by a power of ten by shifting the decimal point) are spotted, making the BCD libraries fast in real world applications.

The following operations may execute as BCD operations:

Addition (+), **Subtraction** (-), **Multiplication** (*)
Performed as a BCD operation when neither operand has a REAL Base Type (both are LONG or DECIMAL) and one has the DECIMAL Base Type. Any digits appearing to the right of 1e31 disappear (wrap), and any to the left of 1e-30 are rounded up.

Division (/) Performed as a BCD operation when neither operand has a REAL Base Type (both are LONG or DECIMAL). Any digits appearing to the right of 1e31 disappear (wrap), and any to the left of 1e-30 are rounded up.

Exponentiation (^) Performed as a BCD operation when the first operand is a DECIMAL or LONG Base Type and the second operand is a LONG Base Type. Any digits appearing to the right of 1e31 disappear (wrap), and any to the left of 1e-30 are rounded.

ABS() Removes the sign from a DECIMAL variable or intermediate value and returns the DECIMAL value.

INT() Truncates a DECIMAL intermediate value and returns a DECIMAL value.

ROUND() If the second parameter is a LONG or DECIMAL Base Type, then rounding is performed as a BCD operation which returns a DECIMAL value. ROUND is very efficient as a BCD operation and should be used to compare REALs to DECIMALs at decimal width.