

# VBIT.DLL

ver. 1.40

---

## VBIT routines for Visual Basic

	AnsiToAscii	Translate from Windows to DOS character set
	AsciiToAnsi	Translate from DOS to Windows character set
	CRLF	Manipulate CR/LF in strings, remove/insert
	Decrypt	Recover string encrypted by Encrypt
	Decrypt7	Recover string encrypted by Encrypt7
	DecryptZ	Recover string encrypted by EncryptZ
	Encrypt	Encrypt a string, make unreadable, linked to a key
	Encrypt7	As Encrypt, but, returns only 7-bit characters
	EncryptZ	As Encrypt, but returns only alphanumeric (A-Z,0-9)
	Find	Find a substring within a string from a given position
NEW	FormNum	Format numbers
	FullPath	Return full path for a file pattern.
	GetDateLong	Convert a "dayNumber&" to a long
	GetDateStr	Convert a "dayNumber&" to a string
	GetDayNumber	Return day number relative to 1/1 1800.
	GetNumDays	Returns number of days between two dates
	Interest	Returns calculated interest in given time period
	LicenseGetCode	For the developers internal use, make license code for applications
	LicenseVBIT	Check for legal license code for VBIT users
	LicenseProgram	Check for legal license code for applications
	LicenseVBIT	Check for legal license code for VBIT users
	Modulus10	Append a CDV (Control Digit Verifier) to number, 10 method
	Modulus10Calc	Return the CDV for a number, 10 method
	Modulus10Valid	Check CDV in number and return false / true, 10 method
	Modulus11	Append a CDV (Control Digit Verifier) to number, 11 method
	Modulus11Calc	Return the CDV for a number, 11 method
	Modulus11Valid	Check CDV in number and return false / true, 11 method
	Num0	Translate from number to string with leading zeros
	Pick	Pick a substring from string
	PickWord	Pick a word from a string
	PickWords	Pick more then one word from a string
	Place	Insert a substring into an other string
	Sound	Play sound
	Strip	Remove a given character from a string
	Subst	Substitute one substring with an other within a string
	SubstAll	Substitute all matching substrings within a string
	SwapChrs	Exchange two characters within a string.
	SwapDate	Exchange positions in a datestring.
	SwapStr	Exchange positions in a string according to a formatted mask
	SysInfo	Return system information as string.
	SysInfoNum	Return system information as integer
	Trace	Write text (string + newline) to debug output window
	TraceStr	Write string to debug output window

**Include VBIT.BAS and VBITTAB.BAS in your projects**

**See also VBITTAB.WRI, VBITVTSS.WRI, VBITFILE.WRI and VBIT.HLP for description of other VBIT functions.**

---

**1996 © InfoTech AS, Bergen, Norway**

---

---

## Function AnsiToAscii

---

Translate string from Windows to DOS character set.

### Usage:

```
Result$ = AnsiToAscii(StringIn$)
```

We want to write some text containing special characters to a DOS file:

### Example:

Open "scan-dos.txt" For Output As #1

```
Write #1, "In Norway and Denmark, we use some special characters:"
Write #1, AnsiToAscii("      [Æ]=[AE], [Ø]=[OE] and [Å]=[AA]")
Write #1, AnsiToAscii("      [æ]=[ae], [ø]=[oe] and [å]=[aa]")
Write #1, AnsiToAscii("In Sweden, they use [Ä] instead of [Æ],")
Write #1, AnsiToAscii("      [ä]=[æ], [Ö]=[Ø] and [ö]=[ø].")
Close #1
```

From DOS, we can look at the file we just made:

**C:\VBIT\TEST> type scan-dos.txt**

In Norway and Denmark, we use some special characters:

```
[Æ]=[AE], [Ø]=[OE] and [Å]=[AA]
[æ]=[ae], [ø]=[oe] and [å]=[aa]
```

In Sweden, they use [Ä] instead of [Æ],

```
[ä]=[æ], [Ö]=[Ø] and [ö]=[ø].
```

If we had not called AnsiToAscii, the result would have looked like this:

In Norway and Denmark, we use some special characters:

```
[Æ]=[AE], [Ø]=[OE] and [Å]=[AA]
[µ]=[ae], [°]=[oe] and [å]=[aa]
```

In Sweden, they use [-] instead of [Æ],

```
[ä]=[µ], [Ö]=[Ø] and [÷]=[°].
```

The message would have lost its meaning because of incompatible character sets.

**See Also:** AsciiToAnsi

---

## Function AsciiToAnsi

---

Translate string from DOS to Windows character set.

### Usage:

```
Result$ = AsciiToAnsi(StringIn$)
```

Read a Dos file to a Windows listbox after proper translation:

### Example:

```
Open "DosFil.Txt" For Input As #1
Do While (Not EOF(1))
    Line Input #1, dostext$
    ListBox.AddItem AsciiToAnsi(dostext$)
Loop
Close #1
```

**See also :** AnsiToAscii

---

## Function CRLF

---

Replace the control character pairs CR (Carriage Return, ascii=13) and LF (Line Feed, ascii=10) with a given character (represented by its ascii value), or the other way around (when value is negative).

This function can be used for translating text files between DOS and UNIX.

The function can be very useful when reading and writing MultiLine TextBoxes in Windows.

### Usage:

```
Result$ = CRLF(StringIn$, asciiValue%)
```

If `asciiValue%` is positive, then all CR/LF character pairs in `StringIn$` will be replaced with the character represented by `asciiValue%` and returned in `Result$`.

When `asciiValue%` is negative, all the occurrences of `Chr$(-asciiValue%)` in `StringIn$` will be replaced with CR/LF and returned in `Result$`.

Simple methode for adding several lines to a MultiLine TextBox:

```
MText1 = CRLF("Line1@Line2@Line3", -Asc("@")) ' Replace "@" with CR/LF
```

Read MultiLine TextBox and convert linefeeds to space:

```
Text1 = CRLF(MText1, Asc(" ")) ' -> "Line1 Line2 Line3"
```

### Example:

Convert file from UNIX format to DOS format (VERY FAST):

```
Sub UnixToDos (ByVal FromFile$, ByVal ToFile$)
    BytesToRead& = FileLen(FromFile$)
    If FileLength(ToFile$) > 0 Then Kill (ToFile$) 'see ITabDir sample
    Open FromFile$ For Input As #1
    Open ToFile$ For Binary Access Write As #2
    Const maxBuff& = 30000 'Read up to 30000 bytes each time
    Do While BytesToRead& > 0
        BuffSize& = BytesToRead&
        If BuffSize& > maxBuff& Then BuffSize& = maxBuff&
        buffer$ = CRLF(Input$(BuffSize&, #1), -10) 'convert LF to CR/LF
        Put #2, , buffer$
        BytesToRead& = BytesToRead& - BuffSize&
    Loop
    Close #1
    Close #2
End Sub
```

---

## Function Decrypt

---

Decrypt a string encrypted by Encrypt.

**Usage:**

```
Result$ = Decrypt(EncryptedText$, EncryptionKey$)
```

**Example:**

```
'Crypt$ => "<'srogjaågkw4åfkae5g0+wk4r935283592+r qawæsqq"  
Secretkey$="MyCode"  
DecryptedString$ = Decrypt(Crypt$, secretkey$)  
'=> DecryptedString$ = "This is the secret text which shall be encrypted"
```

**See also:** DeCrypt7 , DeCryptZ , EnCrypt , EnCrypt7 , EnCryptZ

---

## Function Decrypt7

---

Decrypt a string encrypted by Encrypt7.

**Usage:**

```
Result$ = Decrypt7(EncryptedText$, EncryptionKey$)
```

**See also:** Encrypt7

---

## Function DecryptZ

---

Decrypt a string encrypted by EncryptZ.

**Usage:**

```
Result$ = DecryptZ(EncryptedText$, EncryptionKey$)
```

**See also:** EncryptZ

---

## Function Encrypt

---

Encrypt a string. Will return 8-bit characters without control characters.

**Usage:**

```
Result$ = Encrypt(TextIn$, EncryptionKey$)
```

**Example:**

```
TextIn$= "This is the secret text which shall be encrypted"  
Secretkey$="MyCode"  
Crypt$ = Encrypt(TextIn$, SecretKey$)
```

**See also:** Decrypt , DeCrypt7 , DeCryptZ , EnCrypt7 , EnCryptZ

---

## Function Encrypt7

---

Encrypt a string. Will return only characters from 7 bit ascii values (no control characters).

**Usage:**

```
Result$ = Encrypt7(TextIn$, EncryptionKey$)
```

**See also:** Decrypt7, Encrypt, EncryptZ

---

## Function EncryptZ

---

Encrypt a string. Will return only folded letters (A .. Z) and/or digits (0 .. 9).

**Usage:**

```
Result$ = EncryptZ(TextIn$, EncryptionKey$)
```

**See also:** DecryptZ, Encrypt, Encrypt7

---

## Function Find

---

Search for a substring within an other string from the given position. The position of the found substring is returned, else 0. (In nature equal to the function InStr in Basic). There is no difference between Find and InStr, however it is somewhat easier to remember Find than InStr, don't you think ?

**Usage:**

```
Result% = Find(subString$, inString$, Pos%)
```

**Example:**

```
Instring$ = "12345@@67890"
Pos% = Find("5@", Instring$, 1) ' Pos => 5
Pos% = Find("@", Instring$, 1)  ' Pos => 6
Pos% = Find("@", Instring$, 6)  ' Pos => 7
```

---

## Function FormNum

---

Format number with/round up/down, right justify, 1000-delimiter, adding string in front of number

**Usage:**

```
string$ = FormNum$(number#, decimal%, length%, delimiter$)
```

```
number#      : Number to format (type Double)
decimal%     : number of decimal places
length%      : length on string$
               (ignored if you don't want 1000-delimiter)
delimiter$   : string containing 3 delimiters in row:
               1) String to fill in front of number (typical blank/space).
               2) String for 1000-delimiter
               3) String for decimal-delimiter
```

**Example:**

```
String$ = FormNum(tall1#, 2 ,16, " ,.")
String$ => "          12,345.00"
```

```
String$ = FormNum(tall1#, 2 ,16, " .,")
String$ => "          12.345,00"

String$ = FormNum(tall1#, 0 ,16, " ,.")
String$ => "          12,345"

String$ = FormNum(tall1#, 0 , -16, " ,.")
String$ => "          12345"

String$ = FormNum(tall1#, 0 , -16, "*,.")
String$ => "*****12345"

String$ = FormNum(tall1#, 2 , 16, "*,.")
String$ => "*****12 345.00"

String$ = FormNum(tall1#, 0 , -16, "0,.")
String$ => "00000000000012345"
```

---

## Function FullPath

---

Return full path for a file pattern. The full path will include drive and all directory names for the given pattern.

### Usage:

```
Result$ = FullPath(filePattern$)
```

### Example:

```
' Assume current directory is "C:\VBIT\SAMPLE\TEST":
path$ = FullPath("*.BAS")           '-> "C:\VBIT\SAMPLE\TEST\*.BAS"
path$ = FullPath("../lib/*.DLL")     '-> "C:\VBIT\SAMPLE\LIB\*.DLL"
path$ = FullPath("../VBIT*.WRI")     '-> "C:\VBIT\SAMPLE\VBIT*.WRI"
path$ = FullPath("../..\*.*)"        '-> "C:\VBIT\*.*"
```

---

## Function GetDateLong

---

Convert a "dayNumber&" (returned from GetDayNumber) to a long on the format "yyyymmdd".

### Usage:

```
dateAsLong& = GetDateLong (dayNumber&)
```

### Examples:

```
ldate& = GetDateLong(date1&)      ' ldate&=19951224
ldate& = GetDateLong(date1&+7)    ' ldate&=19951231
```

---

## Function GetDateStr

---

Convert a "dayNumber&" (returned from GetDayNumber) to a string on the format given by dateFmt\$.

### Usage:

```
dateString$ = GetDateStr (dateNum&, dateFmt$)
```

### Examples:

```
sdate$ = GetDateStr(date1&,"DDMMYYYY") ' sdate$="24121995"
```

### Examples:

```
ldate& = GetDateLong(date1&)      ' ldate&=19951224
ldate& = GetDateLong(date1&+7)    ' ldate&=19951231
```

---

## Function GetDayNumber

---

Return day number relative to 1/1 1800. The number returned from this routine can be used for representing dates in a form suitable for calculating number of days between two dates.

### Usage:

```
dayNumber& = GetDayNumber (dateStr$, dateFmt$)
```

### Examples:

```
date1& = GetDayNumber ("24/12-1995", "DD MM YYYY")
date2& = GetDayNumber ("1996 01 01", "YYYY MM DD")
diff& = date2& - date1&           ' should give 8
```

---

## Function GetNumDays

---

Returns number of days between two dates. Valid results for dates from September 14th 1752 to December 31 9999.

### Usage:

```
GetNumDays& (fromDate$, toDate$, dateFormat$, type%)
```

#### fromDate\$:

String containing date "MM","DD","YYYY".  
Position in string is determined by dateFormat\$  
(similar to the function SwapStr\$).

#### toDate\$:

String containing date "DD", "MM" and "YYYY", as described above.



**dateFormat\$:**

String containing the characters "DD", "MM" and "YYYY", where "DD" indicates the position of the day, "MM" the month and "YYYY" the year.

**type%:**

IT_MONTH	' Actual number of days pr month
IT_MONTH_30	' 30 days per month (31st ignored and ' february is also counted as 30 days)

**Example**

```
days1&=GetNumDays("01011995","01031995","DDMMYYYY",IT_MONTH)
days2&=GetNumDays("01011995","01031995","DDMMYYYY",IT_MONTH_30)
' days1& will be 59 and days2& will be 60

n1&=GetNumDays("01-01-1995","03-01-1995","MM-DD-YYYY",IT_MONTH)
' n1& will be the same as days1&
```

---

## Function Interest

---

Returns calculated interest in given time period based on amount and interest rate

Valid results for dates from September 14th 1752 to December 31 9999.

### Usage:

Interest# (fromDate\$, toDate\$, dateFormat\$, amount#, rate#, type%)

#### fromDate\$:

String containing date "MM","DD","YYYY".  
Position in string is determined by dateFormat\$  
(similar to the function SwapStr\$).

#### toDate\$:

String containing date "DD", "MM" and "YYYY", as  
described above.

#### dateFormat\$:

String containing the characters "DD", "MM" and "YYYY",  
where "DD" indicates the position of the day, "MM" the  
month and "YYYY" the year.

#### amount#:

The amount subject to interest calculation.

#### rate#:

The interest rate given in percent.

#### type%:

IT\_MONTH ' Use actual number of days pr month  
IT\_MONTH\_30 ' 30 days per month (31st ignored and  
' february is also counted as 30 days)

+

IT\_YEAR\_360 ' 360 days per year  
IT\_YEAR\_365 ' 365 days per year (also for leap year)  
IT\_YEAR ' Use actual number of days pr year  
' (if start date is in a leap year: 366)

#### Add types for month and year:

IT\_MONTH\_30+IT\_YEAR\_360 ' 30 days per month,  
' 360 days per year

IT\_MONTH+IT\_YEAR\_365 ' Actual number of days,  
' 365 days per year

IT\_MONTH+IT\_YEAR ' Actual number of days,  
' 365/366 days per year  
' (if start date is in a  
' leap year, use 366)

**Example:**

```

loan# = 100000.0
irate# = 10.0 ' interest rate in %
fmt$ = "DD MM YYYY"
typ% = IT_MONTH + IT_YEAR_365
i1#=Interest("01 01 1995","01 01 1996",fmt$,loan#,irate#,typ%)
i2#=Interest("01 01 1995","01 07 1995",fmt$,loan#,irate#,typ%)
i3#=Interest("01 07 1995","01 01 1996",fmt$,loan#,irate#,typ%)
i4#=Interest("01 01 1996","01 07 1996",fmt$,loan#,irate#,typ%)
' i1# = 10000.0 ' one year (365 days)
' i2# = 4958.9 ' 1st half (181 days)
' i3# = 5041.1 ' 2nd half (184 days)
' i4# = 4986.3 ' 1st half next year (182 days: leap year)

typ% = IT_MONTH_30 + IT_YEAR_360
i1#=Interest("01 01 1995","01 01 1996",fmt$,loan#,irate#,typ%)
i2#=Interest("01 01 1995","01 07 1995",fmt$,loan#,irate#,typ%)
i3#=Interest("01 07 1995","01 01 1996",fmt$,loan#,irate#,typ%)
i4#=Interest("01 01 1996","01 07 1996",fmt$,loan#,irate#,typ%)
' i1# = 10000.0 ' one year (360 days)
' i2# = 5000.0 ' 1st half (180 days)
' i3# = 5000.0 ' 2nd half (180 days)
' i4# = 5000.0 ' 1st half next year (180 days)

```

---

## Function LicenseGetCode

---

This function is meant to be used in a stand-alone program and the purpose is to generate licence code for applications. See function [LicenceProgram](#).

**Usage:**

```
Result$ = LicenseGetCode(Name$ , Key$)
```

**Example:**

```
Code$ = LicenseGetCode("Bjorn Nornes", "Key_key_key_1")
```

**See also:** LicenseVBIT , LicenseProgram

---

## Function LicenseProgram

---

This function must be placed in the start-form of your application. If the code and the key is matching,. the function returns True(-1) else False(0). See also function LicenseGetCode.

**Usage:**

```
Result% = LicenseProgram(CustomerName$, Code$, Key$)
```

**Example:**

```
Status% = LicenseProgram("InfoTrade AS", "ABXY12", "Key_Key_Key_1")
```

**See also:** LicenseGetCode , LicenseVBIT

---

## Function LicenseVBIT

---

The buyer of this product will receive a code from InfoTech AS. This will make him/her a registered user of the product and he/she can use the product freely in his/her system.

The table functions are protected by a code for those who have not bought the product. In Visual Basic runmode the protection is in a mild form. When an exe file is made the protection becomes more aggressive and will more often remind the user of the lack of payment. Despite this, the user can fully test the product or use the 'free to use functions in the package.

**Usage:**

```
Result% = LicenseVBIT(Name$ , Code$)
```

Result% will contain a True(-1) if a legal code is given, else False(0).

**Example:**

```
Status% = LicenseVBIT("Douglas Moore", "TT4LBT")
```

**See also:** LicenseGetCode , LicenseProgram

---

## Function Modulus10

---

Append a Control Digit Verifier to the input string based on the modulus 10 formula . All characters except digits in the StrIn\$ are ignored during calculation.

**Usage:**

```
Result$ = Modulus10(StrIn$)
```

**Example:**

```
CustNum$ = Modulus10("95101201230") ' CustNum$ = "951012012302"
```

**See also:** Modulus11 , Modulus10Calc , Modulus11Calc , Modulus10Valid , Modulus11Valid\_

---

## Function Modulus11

---

Append a Control Digit Verifier to the input string based on the modulus 11 formula. All characters except digits in the StrIn\$ are ignored during calculation.

**Usage:**

```
Result$ = Modulus11(StrIn$)
```

**Example:**

```
Account$ = Modulus11("9521.05.6932") ' Account$ = "9521.05.69325"
```

**See also:** Modulus10 ,Modulus10Calc ,Modulus11Calc ,Modulus10Valid ,Modulus11Valid

---

## Function Modulus10Calc

---

The function returns a control digit based on CDV modulus 10 calculation over the StrIn\$.

**Usage:**

```
Result$ = Modulus10Calc(StrIn$)
```

**Example:**

```
CD$ = Modulus10Calc("95101201230") ' CD$ = "2"
```

**See also:** Modulus10 , Modulus11 , Modulus11Calc , Modulus10Valid , Modulus11Valid

---

## Function Modulus11Calc

---

The function returns a control digit based on CDV modulus 11 calculation over the StrIn\$.

**Usage:**

```
Result$ = Modulus11Calc(StrIn$)
```

**Example:**

```
CD$ = Modulus11Calc("9521.05.6932") ' CD$ = "5"
```

**See also:** Modulus10 , Modulus11 , Modulus10Calc , Modulus10Valid , Modulus11Valid

---

## Function Modulus10Valid

---

The function returns True(-1) if the last character of StrIn\$ is a valid CDV based on the modulus 10 formula, else it returns False(0).

**Usage:**

```
Result% = Modulus10Valid(StrIn$)
```

**Example:**

```
If Modulus10Valid("9521.05.69325") Then Status="OK"
```

**See also:** Modulus10 , Modulus11 , Modulus10Calc , Modulus11Calc , Modulus11Valid

---

## Function Modulus11Valid

---

The function returns True(-1) if the last character of StrIn\$ is a valid CDV based on the modulus 11 formula, else it returns False(0).

**Usage:**

```
Result% = Modulus11Valid(StrIn$)
```

**Example:**

```
If Not Modulus11Valid("9521.05.69328") Then Status="ERROR"
```

**See also:** Modulus10 , Modulus11 , Modulus10Calc , Modulus11Calc , Modulus10Valid\_

---

## Function Num0

---

Convert a positive number to a string with leading zeros.  
The number of digits must be given in the call, max 9.

**Usage:**

```
Result$ = Num0 (Number&, Digits%)
```

**Example:**

```
String$ = Num0 (1,3)      => "001"
String$ = Num0 (1234,9)   => "000001234", max number of digits.
String$ = Num0 (1234,10)  => "1234"
```

---

## Function Pick

---

Pick one or more characters from a text string. The position of the first character, and the wanted number of characters from that position must be given in the call. The function returns a string.

**Usage:**

```
Result$ = Pick (StringIn$, FromPos%, Length%)
```

Requiring more characters than the input sting contains, causes the function to fill the surplus characters with blanks.

If the wanted number of characters is set to 0, the function will return rest of the string from the given position.

If the position is given as a negative number, the start position will be relative to the end of the string. -1 is the last position in the string, -2 is the last but one, and so on. 0 as position will be interpreted as the position after the last character.

If the number wanted is given as a negative number, the routine will pick characters from the left of the given position, inclusive.

**Example:**

```
String$ ="Example of the Pick function in use"
Result$ = Pick (String$,1,7)    'Result$ => "Example"
Result$ = Pick (String$,32,11)  'Result$ => " use          "
Result$ = Pick (String$,32,0)   'Result$ => " use"
Result$ = Pick (String$,-10,8)  'Result$ => "ion in u"
Result$ = Pick (String$,8,-6)   'Result$ => "ample "
Result$ = Pick (String$,-5,-2)  'Result$ => "in"
Result$ = Pick (String$,0,-3)   'Result$ => "se "(0 is the pos. after last
                                char)
```

**See also:** PickWord , PickWords

---

## Function PickWord

---

Pick a word from a string. Declaring the position number of the wanted word and the delimiter, the function returns the wanted word as a string.

### Usage:

```
Result$ = PickWord(StringIn$, WordNumber%, Delimiter%)
```

The delimiter must be given as an ascii value. For the purpose of increasing the readability the VB function "Asc( )" can be used. Given semicolon as delimiter: Asc(";"). Having a do-while-loop where PickWord will be called many times, it would enhance speed to initialize a variable outside the loop: Semicolon% = Asc(";")

Ignoring leading delimiters and /or deal with them as one connected delimiter, the negative ascii value for the delimiter should be given: Semicolon% = -Asc(";")

### Example:

```
text$ = "Here;is;an;;example;using PickWord"      'Result

Result$ = PickWord(text$, 3, Asc(";"))             '"an"
Result$ = PickWord(text$, 5, Asc(";"))             '"example"
Result$ = PickWord(text$, 6, 59)                   '"using PickWord"
Result$ = PickWord(text$, 5, -59)                   '"using PickWord"
Result$ = PickWord(text$, 2, 32)                    '"PickWord"
Result$ = PickWord(text$, 2, Asc("e"))              '"r"
```

**See also:** Pick , PickWords\_

---

## Function PickWords

---

Pick more than one word from a string. If you only need one word, you ought to use PickWord.

### Usage:

```
Result$ = PickWords(StringIn$, WordNumber%, NumWanted%, Delimiter%)
```

Given the word number for the first word in the string and the number of wanted words, the function returns a string.

In order to get all words from a given wordnumber, 0 as number must be used.

The delimiter must be given as an ascii value. For the purpose of increasing the readability the VB function "Asc( )" can be used. Given semicolon as delimiter: Asc(";"). Having a do-while-loop where PickWord will be called many times, it would enhance speed to initialize a variable outside the loop: Semicolon% = Asc(";")

Ignoring leading delimiters and /or deal with them as one connected delimiter, the negative ascii value for the delimiter should be given: Semicolon% = -Asc(";")

### Example:

```
text$ = ";Here;is;an;;example;using PickWords"    'Result
Result$ = PickWords(text$, 3, 2, Asc(";"))         '"is;an"
Result$ = PickWords(text$, 3, 3, Asc(";"))         '"is;an"
Result$ = PickWords(text$, 4, 3, Asc(";"))         '"an;;example"
Result$ = PickWords(text$, 3, 2, -Asc(";"))        '"an;example"
Result$ = PickWords(text$, 2, 0, Asc(" "))         '"PickWords"
' note the leading ";" in text$
```

**See also:** Pick , PickWord\_

---

## Function Place

---

Superimpose a string on a copy of "tostring" in the given position and return the result as a string. If one want the whole "fromstring" one can use 0 as the number of wanted characters, else use the actual number of wanted characters picked from "fromstring". If the given number is greater then the length of the "fromstring", the function will fill the surplus number by space.

**Usage:**

Result\$ = Place(FromString\$, ToString\$, Pos%, Length%)

**Example:**

tostring\$ = "*****"	'Result
Result\$ = Place("TEST", tostring\$, 4, 0)	'****TEST****
Result\$ = Place(" TEST", tostring\$, 3, 6)	'*** TEST **'
Result\$ = Place("TEST", tostring\$, 1, 2)	'TE*****'
Result\$ = Place(Num0(123,6), tostring\$, 7, 0)	'*****000123'

---

## Function Sound

---

Play sound through PC-speaker or through sound-card !

Sound "+"	' OK signal (same as Beep)
Sound "?"	' System sound for Question
Sound "!"	' System sound for Exclamation (error)
Sound "*"	' System sound for Asterisk ("finished")
Sound "."	' System sound for Critical Stop
Sound "-"	' PC speaker beep
Sound "FILENAME.WAV"	' Play WAV-file. If the file is not found in ' the current/given directory, the routine will ' look for the file in the WINDOWS directory.

---

## Function Strip

---

Remove a given character from a string.

**Usage:**

Result\$ = Strip(StringIn\$, Char\$, Type%)

**Type:**

STRIP_L	Remove leading delimiters, (as LTRIM i Basic)
STRIP_T	Remove trailing delimiters, (as RTRIM i Basic)
STRIP_LT	Remove leading and trailing delimiters,(as TRIM i Basic)
STRIP_ALL	Remove all delimiters

What sets Strip and VB's \*TRIM funksjon apart, is that Strip may remove any character where \*TRIM only removes "space".

**Example:**

String\$ = "***T*E*S*T***"	
Result\$ = Strip(String\$, "*", STRIP_L)	' "T*E*S*T***"
Result\$ = Strip(String\$, "*", STRIP_T)	' "***T*E*S*T"
Result\$ = Strip(String\$, "*", STRIP_LT)	' "T*E*S*T"
Result\$ = Strip(String\$, "*", STRIP_ALL)	' "TEST"



If you want to remove repeating embedded delimiters, the function PickWords can be suitable.

```
String$ = ";;This;;is;an;;;example;using;;PickWords;;"
Result$ = PickWord(String$, 1, 0, -Asc(";"))
'Result$ : "This;is;an;example;using;PickWords"
```

---

## Function Subst

---

Exchange a substring with another string from a given position in the third string and return the resultstring. The position must be given as a variable. The variable will be changed by the function. Into this variable the next position is given if there are more than one occurrence of the substring in the instring after the position, else a zero will be returned. The returned position will be related to the resultstring. Search for inString\$ is case sensitive.

Usage:

```
Result$ = Subst(OldStr$, NewStr$, inString$, Pos%)
```

This call will change the variable Pos%.

### Example

```
pos%=1                                'startpos for searching in the
                                      'instring$

Inn$= "5 hours a kr 100: kr 500"
Res$= Subst("kr", "NOK", Inn$, pos%) 'Res$ : "5 hours a NOK 100: kr 500
                                      'pos% : 20 to next occurrence

Res$= Subst("kr", "NOK", Inn$, pos%) 'Res$ : "5 hours a NOK 100: NOK 500
                                      ' pos% : 0
```

See also: SubstAll

---

## Function SubstAll

---

Exchange all the occurrences of oldstring\$ with newstring\$ in a copy of the instring\$ which is returned as a result. Search for inString\$ is case sensitive.

Usage:

```
Result$ = SubstAll(OldStr$, NewStr$, inString$)
```

### Example

```
res$ = SubstAll("1 ", "@@", "1111 222221 33333 444441 555555")
res$ = "111@@22222@@33333 44444@@555555"
res$ = SubstAll("is", "was", "This is an example")
res$ = "Thwas was an example"
```

See also: Subst

---

## Function SwapChrs

---

Swap two characters within a string. The argument "Character" contains the two characters which are to be swapped. The function returns a string where all the occurrences of the specified characters are swapped. A typical example would be to swap the characters period(.) and comma(,).

**Usage:**

```
Result$ = SwapChrs(String$, Characters$)
```

**Example:**

```
Result$ = SwapChrs("1.234.567,00", ".,") '=> "1,234,567.00"
```

**See also:** SwapDate , SwapStr

---

## Function SwapDate

---

Swap the position of the year and day within a datestring with format "YYMMDD" or "DDMMYY"

The function SwapStr may be used as replacement for SwapDate. Please refer to SwapStr

**Usage:**

```
Result$ = SwapDate(Date$)
```

**Example:**

```
NewDate$ = SwapDate("241294") ' => "941224"
```

```
NewDate$ = SwapDate("941224") ' => "241294"
```

**See also:** Swap , SwapStr

---

## Function SwapStr

---

This function can replace SwapDate, but can also be used in other occasions. The "fromFmt\$" and the "toFmt\$" consist of letters which describe the wanted formate. E.g. "DD-MM-YY", "YYMMDD", (Year, Month, Day).

**Usage:**

```
Result$ = SwapStr(StrIn$, FromFmt$, ToFmt$)
```

Letters which are found in both FromFmt\$ and the ToFmt\$ give the position and length, repeating equal letters, of the string which to be picked from "StrIn\$" and placed in the Result\$. The ToFmt\$ is the template for the Result\$. All positions which are not overwritten will be left in the Result\$ untouched. If the length of the substring FromFmt\$ is less then the lenght of the ToFmt\$ , leading zeros will be put into the Result\$. If the length of the substring ToFmt\$ is less then the length of FromFmt\$ then the function picks the number of characters from the left which can be placed according the template. E.g. 1994 (yy) => 94.

**Example:**

```
ResultString$ = SwapStr("241294", "ddmmyy", "yymdd") "941224"
```

```
ResultString$ = SwapStr("941224", "yymdd", "dd/mm-yy") "24/12-94"
```

```
ResultString$ = SwapStr("12-24-1994", "mm dd yyyy", "ddmmyy") "241294"
```

**See also:** Swap SwapDate

---

## Function SysInfo

---

This returns system information about the PCs environment as string.

### Usage:

```
Result$ = SysInfo(What%)
```

### What%

```
SCREEN_SIZE_X
SCREEN_SIZE_Y
SCREEN_SIZE_PALETTE
MEMORY_FREE_KB
MEMORY_BIGGEST_FREE_BLOCK_KB
```

### Result\$

```
The width of the screen
The height of the screen
The number of colors available
Free memory measured in KiloBytes
Biggest free memory block measured in
KiloBytes

DISK_DRIVE           Current drive, (1="A", 2="B", 3="C" .... )
DISK_FREE_KB         Free disk space measured in KiloBytes
DISK_SIZE_KB         Total disk space measured in KiloBytes
DISK_TYPE            Drive type (see below)
```

The following is only defined for SysInfo (string only):

```
DIR_WINDOWS          Current path for the \WINDOWS\directory
DIR_WINDOWS_SYSTEM   Current path for the \WINDOWS\SYSTEM\ directory
DISK_PATH            Current D:\DIRECTORY\NAME
DISK_VOLUME_LABEL    Disk label, (name, 11 char.)
DISK_VOLUME_DATE     Volume label date "YYYYMMDD"
DISK_VOLUME_TIME     Volume label time "TT:MM:SS"
```

DISK\_TYPE returns

```
"REMOVABLE"
"FIXED"
"REMOTE"
"?"
```

For all "DISK\_...." parameters, the current disk drive will be used unless a disk drive is specified.

Specifying an other drive goes as follows:

Add the drive number or the ascii value of the drive letter to the argument (What%).

### Examples

```
si$ = SysInfo(DISK_SIZE_KB + 1)      '=> Regarding drive A
si$ = SysInfo(DISK_PATH_KB + 2)      '=> Regarding drive B
si$ = SysInfo(DISK_SIZE_KB + Asc("A")) '=> Regarding drive A
si$ = SysInfo(DISK_FREE_KB + Asc("C")) '=> Regarding drive C
```

**See also:** SysInfoNum

---

## Function SysInfoNum

---

This returns system information about the PCs environment. SysInfoNum as long integer when possible.

### Usage:

Result& = SysInfoNum(What%)

### What%

SCREEN\_SIZE\_X  
 SCREEN\_SIZE\_Y  
 SCREEN\_SIZE\_PALETTE  
 MEMORY\_FREE\_KB  
 MEMORY\_BIGGEST\_FREE\_BLOCK\_KB  
 DISK\_DRIVE  
 DISK\_FREE\_KB  
 DISK\_SIZE\_KB  
 DISK\_TYPE

### Result&

The width of the screen  
 The height of the screen  
 The number of colors available  
 Free memory measured in KiloBytes  
 Biggest free memory block measured in KiloBytes  
 Current drive, (1="A", 2="B", 3="C" .... )  
 Free disk space measured in KiloBytes  
 Total disk space measured in KiloBytes  
 Drive type (see below)

DISK\_TYPE returns

DRIVE\_REMOVABLE  
 DRIVE\_FIXED  
 DRIVE\_REMOTE  
 0

For all "DISK\_...." parameters, the current disk drive will be used unless a disk drive is specified.

Specifying an other drive goes as follows:

Add the drive number or the ascii value of the drive letter to the argument (What%).

### Example:

si& = SysInfoNum(DISK\_SIZE\_KB + Asc("D")) '=> Regarding drive D

**See also:** SysInfo

---

## Sub Trace

---

Output a line of text followed by a linefeed to the debug output device. The debug output device can be a secondary monochrome screen, a screen connected to a Com-port or a window on the screen. You have to run a special program for activating the debug device. A suitable program for this purpose is DBWIN.EXE.

### Usage:

Trace debugText\$

This routine together with TraceStr is a good alternative to the standard debug in Visual Basic.

It can be used for dumping contents of variables, tracing events etc.

### Example:

```
Trace "Click event: Mouse button=" & Button & ", X=" & X & ", Y=" & Y
' output: Click event: Mouse button=1, X=12, Y=43
```

**See also :** TraceStr

---

## Sub TraceStr

---

Output a text string to the debug output device.

**Usage:**

TraceStr debugText\$

**Example:**

```
TraceStr "Click event: Mouse button="
TraceStr Button
TraceStr ", X=" & X
Trace ", Y=" & Y      ' terminate line.
' output: Click event: Mouse button=2, X=122, Y=143
```

**See also :** Trace