Visual Basic Invisible Tools v1.40

What is a Table?

A VBIT table is a dynamic array or matrix of variable length text strings.

It is possible to define arrays of strings in native Visual Basic too, but lacks flexibility, and the practical limits makes it completely useless compared to tables in VBIT:

```
Static SmallMatrix(500, 10) As String ' Standard Visual Basic
For i% = 1 To 500
    For j% = 1 To 10
        SmallMatrix(i%, j%) = "TESTING"
    Next j%
Next i%
```

"OUT OF STRING SPACE" is the message from Visual Basic !
I´ve got 16 MB of RAM, but VB does not want to use it.

String Arrays in VB are not going to be mentioned any more, from this point. In the following, the term Array will only be used for describing a table with only one column.

```
' The good news are: the following works fine

BigTable& = ITabNew(5000, 10) ' VBIT 10 times bigger - no problem !
 For i% = 1 To 5000
    For j% = 1 To 10
        ITabPut BigTable&, i%, j%, "TESTING"
    Next j%
 Next i%

'You may put several megabytes of data into the VBIT tables.
```

The powerful routines for table searching, sorting, file I/O and all the other routines found in VBIT will open a new world for all Visual Basic programmers - giving the programming the power you have dreamed of !

Where to find VBIT?

The latest release of VBIT will always be avaliable at Trader's Mascot BBS

This is a BBS running Excalibur : Phone : +47 7012 9014

You may also find VBIT on InterNet

http://www.prodat.no/infotech/

Important information for IdbTools users

From version 1.15, this tool is called VBIT.
Users of previous versions (IdbTools) should be aware of the following changes:

Old routine name	**New routine name**	**Comments**
LicenseIdbTools	**LicenseVBIT**	Old license codes will still be valid
IdbTrace	**Trace**	
IdbTraceStr	**TraceStr**	
VersionIdbTools	**FileGetVersion("VBIT.DLL")**	The new routine will work for other files too

The following calls will be slightly different from previous versions:

SysInfo(DIR_WINDOWS)
SysInfo(DIR_WINDOWS_SYSTEM)
SysInfo(DISK_PATH)

The returned path will always be terminated by "\" (e.g. "C:\WINDOWS\"). **VBIT.BAS** should be included in your projects instead of IDBTOOLS.BAS and IDBTABLE.BAS

If you use the VTSS calls, **VBITVTSS.BAS** should be included too

Strip function is changed, the second parameter should now be a string.

Result$ = **Strip**(StringIn$, **Char$**, Type%).

A

AnsiToAscii
AsciiToAnsi

C

CRLF

D

Decrypt
Decrypt7
DecryptZ

E

Encrypt
Encrypt7
EncryptZ

F

FileExist
FileFindPath
FileGetAttr
FileGetDate
FileGetExt
FileGetFileName
FileGetPath
FileGetSize
FileGetTime
FileGetVersion
Find
FormNum
FullPath

G

GetDateLong
GetDateStr
GetDayNumber
GetNumDays

I

IniFileGetString
IniFilePutString
Interest
ITabBlankLine
ITabBlankLines
ITabCopy
ITabCopyDataToVTSS
ITabCopyFromVTSS
ITabCopyToGrid
ITabCopyToVTSS
ITabDelete
ITabDir
ITabEnvList
ITabEnvString
ITabFastSort

ITabFind
ITabFindGE
ITabFromString
ITabGet
ITabGetColWidth
ITabGetInt
ITabGetLine
ITabGetLong
ITabGetNumColumns
ITabGetNumLines
ITabGetReal
ITabGetSize
ITabInsertLine
ITabInsertLines
ITabNew
ITabNewArray
ITabPut
ITabPutInt
ITabPutLine
ITabPutLong
ITabPutReal
ITabRead
ITabReadFixedRecLenFile
ITabReadFmt
ITabRemoveLine
ITabRemoveLines
ITabSetMaxDecimalsFromVTSS
ITabSmartSort
ITabToVTSS
ITabWrite

L

LicenseGetCode
LicenseProgram
LicenseVBIT

M

Modulus10
Modulus10Calc
Modulus10Valid
Modulus11
Modulus11Calc
Modulus11Valid

N

Num0

P

Pick
PickWord
PickWords
Place

S

Strip
Subst
SubstAll
Sound

VBIT File Routines

FileExist	Function to check if a file exist
FileFindPath	Sarches for a given filename, returns path
FileGetAttr	Returns the attributes for a file as string
FileGetDate	Returns date for a file
FileGetExt	Returns extension of a complete filename.
FileGetFileName	Returns filename without extention
FileGetPath	Returns PATH-part of a complete filename
FileGetSize	Returns filesize as Long
FileGetTime	Returns the time for a file as string
FileGetVersion	Get the current version number of VBIT
IniFileGetString	Read data from a given address in an INI-file
IniFilePutString	Write data to a given address in an INI-file

Function FileExist

Example

Back

Function to check if a file exist. FileExist returns (True/False)

Usage:

```
Function FileExist%(fileName$)
```

Example FileExist

```
AutoTab& = ITabRead("C:\AUTOEXEC.BAT",IT_TEXTFILE+IT_ASCII) ' edit file......
' Write back, but keep original file as ".BAK": ITabWrite(autoTab&,"C:\
AUTOEXEC.$$$",IT_TEXTFILE+IT_ASCII) If FileExist("C:\AUTOEXEC.BAK") Then
Kill "C:\AUTOEXEC.BAK"
End If
Name "C:\AUTOEXEC.BAT" As "C:\AUTOEXEC.BAK"
Name "C:\AUTOEXEC.$$$" As "C:\AUTOEXEC.BAT"
ITabDelete autoTab&
```

Function FileFindPath

Example

Back

This function searches for a given filename, and returns the filename complete with path.

The search is first performed in the current path, if the file is not found the search continues in the \ WINDOWS\SYSTEM directory, and finally in the PATH settings from environment.

Usage :

```
f_file$ =  FileFindPath$(file$)
```

Example FileFindPath

```
file$ =  FileFindPath$("VBIT.DLL")

'Result : file$: "C:\WINDOWS\SYSTEM\VBIT.DLL"
```

Function FileGetAttr

Back

Returns the attributes for a file as string. Format : "ADHRS"

A: Archive (set when file is changed - used by back-up systems) D: Directory name
H: Hidden file
R: Read-Only file
S: System file

Usage:

```
file$ = "C:\WINDOWS\SYSTEM\VBIT.DLL"
Attr$ = FileGetAttr(file$)
```

Function FileGetExt

Example

Back

Returns extension of a complete filename.

Usage:
file$ = FileGetExt(file$)

Example FileGetExt

```
fil$ = "C:\WINDOWS\SYSTEM\VBIT.DLL"
f_ext$  = FileGetExt(fil$)          ' f_ext:  "DLL"
f_fil$  = FileGetFilename(fil$)     ' f_fil:  "VBIT"
f_name$ = FileGetName(fil$)         ' f_name: "VBIT.DLL"
f_path$ = FileGetPath(fil$)         ' f_path: "C:\WINDOWS\SYSTEM\"
```

Function FileGetDate

Back

Returns date for a file. Format : "YYYYMMDD"

Usage:

```
file$ = "C:\WINDOWS\SYSTEM\VBIT.DLL"
date$ = FileGetDate(file$)
```

Function FileGetFileName

Back

Back

Returns filename without extention of a complete filename.(D:\path\file.ext)

Usage:

```
f_fil$  = FileGetFilename(fil$)
```

Example FileGetFileName

```
fil$ = "C:\WINDOWS\SYSTEM\VBIT.DLL"
f_fil$  = FileGetFilename(fil$)     ' f_fil:  "VBIT"
```

Function FileGetPath

Back

Back

Returns PATH-part of a complete filename (D:\PATH\FILE.EXT)

Usage:

```
path$ =  FileGetPath$(file$)
```

Example FileGetPath

```
wordpath$ =  FileGetPath$(FileFindPath$("WINWORD.EXE")) 'wordpath$: "D:\
WINWORD\"
```

Function FileGetSize

Back

Returns filesize as Long. If the file does not exist, the return value is 0. (Visual Basic's FileLen causes a run-time error if the file is uavaliable)

Usage:

```
size& = FileGetSize(file$)
```

Function FileGetTime

Back

Back

Returns the time for a file as string. Format : "HHMMSS"

Usage:

```
Time$ = FileGetTime(file$)
```

Example FileGetTime

```
file$ = "WINWORD.EXE"
time$ = FileGetTime(FileFindPath(file$))
```

Function FileGetVersion

Back

Back

Get version number of given file.

If the given file is not found in the given path (or current directory), the routine will search for it in \ WINDOWS\SYSTEM\ and eventually the path given by the current environment.

Usage:
Result$ = FileGetVersion(FileName$)

The version number is returned as String, but it can be converted to a Double.

The version information is obtained from the standard Windows version information found in most files made for Windows, e.g. DLL-, EXE-, DRV- files. If this information is not present in the given file, this routine will make up a version number based on the file's date and time.

The Windows file version information is internally given by two 32-bit numbers, but is usually presented as four 16-bit numbers seperated by periods, e.g. **"3.10.002.5"**. There are also similar information for product version, but this is ignored here.

This routine will return a string which is possible to convert to a number (Double), and will use the following rules:

The version is given by: **"aa.bb.cc.dd"**, where **"aa.bb"** normally is the official version number and **"cc.dd"** is the revision number, if given at all. If the second part is zero, it is ignored by this routine. If it is not zero, the second number will be divided by 1000 and appended to the first number after stripping off the decimal point of the second number. A version number like this,**"3.10.002.5"**, will look like this when returned from this routine: **"3.100025" ("3.10" & ".0025")**.

If the file does not contain any version information, the version number returned from this routine will be made up using the file's date and time information: **"0.0000YYYYMMDDhhmmss"**.

Example FileGetVersion

```
Result$ = FileGetVersion("VBIT.DLL")      ' Result will be "1.20" for this
version
Result$ = FileGetVersion("C:\VB\VB.EXE")  ' Result: "3.000537"
Result$ = FileGetVersion("C:\CONFIG.SYS") ' Result: "0.0000199503101207 48"
(date/time info)
```

Function IniFileGetString

Back

See also

Back

Read data from an INI-file. Filename, section and a profile name is given and the function returns a string containing the profile string. If the profile name do not exist, the return value is an empty string. If the filename is given without any path, the system will start looking for the file in the Windows directory.

Usage:

```
Result$ = IniFileGetString(FileName$, Section$, Name$)
```

The section name must be given without brackets,

Wrong => "[SectionName]"
Correct => "SectionName"

Example IniFileGetString

```
StartProg$ = IniFileGetString("SYSTEM.INI", "boot", "shell")
' Returns perhaps "progman.exe"

String$ = IniFileGetString("WIN.INI", "MS user info", "DefName") ' Return
information about the user from "WIN.INI"
```

See Also

INIFilePutString

Function IniFilePutString

Back

See also

Back

Write data to an INI-file. Given the filename, section, name and the data to be written.

If the filename is given without any path, the system will start looking for the file in the Windows directory . The section name must be given without brackets. The function returns True(-1) if the call was successful, else False(0).

Usage:

```
Result% = IniFilePutString(FileName$, Section$, Name$, Data$)
```

Example IniFilePutString

```
Result%=IniFilePutString("MYPROG.INI", "Licence", "Name", "John Doe")

' Will write within the file "\WINDOWS\MYPROG.INI":
[Licence]
Name = John Doe

OK%=IniFilePutString("WIN.INI", "Desktop", "Wallpaper", "c:\pic\my.bmp")
' This statement will change the wallpaper, taking effect from the next
startup of Windows.
```

See Also

INIFileGetString

VBIT Misc Routines

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
FormNum	Format number
FullPath	Return full path for given 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
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 sounds
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

Function AnsiTiAscii

Back

Back

Back

Translate string from Windows to DOS character set.

Usage:

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

Sample collection

Example AnsiToAscii

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

```
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

Back

Back

Back

Translate string from DOS to Windows character set.

Usage:

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

Sample collection

Example AsciiToAnsi

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

```
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

AnsiAscii

Function CRLF

Back

Back

Replace the control character pairs CR (Carrriage 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 occurences 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"
```

Sample collection

Example CRLF

```
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 Decrypt

```
'Crypt$ => "<'srogjaågkw4åfkae5g0+wk4r935283592+r qawæsqg" 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

Back

Back

Decrypt a string encrypted by Encrypt7.

Usage:

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

See Also

Decrypt
DecryptZ
Encrypt
Encrypt7
EncryptZ

Function DecryptZ

Back

Back

Decrypt a string encrypted by EncryptZ.

Usage:

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

See Also

Decrypt
Decrypt7
Encrypt
Encrypt7
EncryptZ

Function Encrypt

Back

Back

Back

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

Usage:

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

Example Encrypt

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

See Also

Decrypt
Decryot7
DecryotZ
Encrypt7
EncryptZ

Function Encrypt7

Back

Back

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

Usage:

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

See Also

Decrypt
Decryot7
DecryotZ
Encrypt
EncryptZ

Function EncryptZ

Back

Back

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

Usage:

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

See Also

Decrypt
Decryot7
DecryotZ
Encrypt
Encrypt7

Function Find

Back

Back

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).

Usage:

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

Sample collection

Example Find

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

Function FormNum

Back

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 containg 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

Exsample:

```
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$ => "0000000000012345"
```

Function Fullpath

Back

Back

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 FullPath

```
' 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("..\IDBT*.WRI")      '-> "C:\VBIT\SAMPLE\IDBT*.WRI" path$ =
FullPath("..\..\*.*")         '-> "C:\VBIT\*.*"
```

Function GetDateLong

Back

Back

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

Usage:

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

Example FullPath

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

Function GetDateStr

Back

Back

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

Usage:

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

Example FullPath

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

Function GetDayNumber

Back

Back

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$, dateFmt$)
```

Example FullPath

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

Function GetNumDays

Back

Back

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 GetNumDays

```
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

Back

Back

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 Interest

```
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 LicenseGetCode

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

See Also

LicenseProgram
LicenseVBIT

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 LicenseProgram

```
Status% = LicenseProgram("Trader's Mascot AS", "ABXY12", "Key_Key_Key_1")
```

See Also

LicenseGetCode
LicenseVBIT

Function LicenseVBIT

Back

Back

Back

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 exefile 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 LicenseVBIT

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

See Also

LicenseGetCode
LicenseProgram

Function Modulus10

Back

Back

Back

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 Modulus10

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

See Also

Modulus10Calc
Modulus10Valid
Modulus11
Modulus11Calc
Modulus11Valid

Function Modulus10Calc

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

Usage:

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

Example Modulus10Calc

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

See Also

Modulus10
Modulus10Valid
Modulus11
Modulus11Calc
Modulus11Valid

Function Modulus10Valid

Back

Back

Back

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 Modulus10Valid

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

See Also

Modulus10
Modulus10Calc
Modulus11
Modulus11Calc
Modulus11Valid

Function Modulus11

Back

Back

Back

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 Modulus11

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

See Also

Modulus10
Modulus10Calc
Modulus10Valid
Modulus11Calc
Modulus11Valid

Function Modulus11Calc

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

Usage:

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

Example Modulus11Calc

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

See Also

Modulus10
Modulus10Calc
Modulus10Valid
Modulus11
Modulus11Valid

Function Modulus11Valid

Back

Back

Back

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 Modulus10Valid

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

See Also

Modulus10
Modulus10Calc
Modulus10Valid
Modulus11
Modulus11Calc

Function Num0

Back

Back

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 Num0

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

Function Pick

Back

Back

Back

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.

Sample Collection

Example Pick

```
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

Back

Back

Back

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(";")

Sample collection

Example PickWord

```
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

Back

Back

Back

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 PickWords

```
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

Back

Back

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 Place

```
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

Back

Play sound through PC-speaker og 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

Back

Back

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 delmiters, (as RTRIM i Basic)
STRIP_LT	Remove leading and trailing delmiters,(as TRIM i Basic)
STRIP_ALL	Remove all delmiters

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

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

Example Strip

```
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"
```

Remove a given charcter, given as an ascii value, from a string.

Function Subst

Back

Back

Back

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 occurence 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 Subst

```
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

Back

Back

Back

Exchange all the occurences 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 SubstAll

```
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

Back

Back

Back

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 occurences of the specified characters are swapped. A typical example would be to swap the characters period(.) and comma(,).

Usage:

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

Example SwapChrs

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

See Also

SwapDate
SwapStr

Function SwapDate

Back

Back

Back

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 SwapDate

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

See Also

SwapChrs
SwapStr

Function SwapStr

Back

Back

Back

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 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 SwapStr

```
ResultString$ = SwapStr("241294", "ddmmyy", "yymmdd")          '"941224"
ResultString$ = SwapStr("941224", "yymmdd", "dd/mm-yy")        '"24/12-94"
ResultString$ = SwapStr("12-24-1994", "mm dd yyyy", "ddmmyy") '"241294"
```

See Also

SwapChrs
SwapDate

Function SysInfo

Back

Back

This returns system information about the PCs environment as string.

Usage:

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

What%	***Result$***
SCREEN_SIZE_X	The width of the screeen
SCREEN_SIZE_Y	The height of the screeen
SCREEN_SIZE_PALETTE	The number of colors available
MEMORY_FREE_KB	Free memory measured in KiloBytes
MEMORY_BIGGEST_FREE_BLOCK_KB	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)
DISK_CLUSTER_SIZE [+ drive]	
DISK_SECTOR_SIZE [+ drive]	
DISK_SECTORS_PR_CLUSTER [+ drive]	
DISK_AVAILABLE_CLUSTERS [+ drive]	
DISK_TOTAL_CLUSTERS [+ drive]	

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

DIR_WINDOWS	Current path for the \WINDOWS\SYSTEM\ 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"
"CDROM"
"?"

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%).

Note: Number of clusters is limited to &Hffff (16 bit),
so this information wil be incorrect for very big drives,
such as CD-ROMs (most programs, including Windows File Manager,
have the same problem). If the number of clusters is &Hffff and

the drive type is CDROM, SysInfo/SysInfoNum will now return
650000 for type (DISK_SIZE_KB [+ drive]).

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

Back

Back

Back

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

Usage:

```
Result& = SysInfoNum(What%)
```

What%	***Result&***
SCREEN_SIZE_X	The width of the screeen
SCREEN_SIZE_Y	The height of the screeen
SCREEN_SIZE_PALETTE	The number of colors available
MEMORY_FREE_KB	Free memory measured in KiloBytes
MEMORY_BIGGEST_FREE_BLOCK_KB	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)
DISK_CLUSTER_SIZE [+ drive]	
DISK_SECTOR_SIZE [+ drive]	
DISK_SECTORS_PR_CLUSTER [+ drive]	
DISK_AVAILABLE_CLUSTERS [+ drive]	
DISK_TOTAL_CLUSTERS [+ drive]	

DISK_TYPE returns

DRIVE_REMOVABLE
DRIVE_FIXED
DRIVE_REMOTE
DRIVE_CDROM
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%).

Note: Number of clusters is limited to &Hffff (16 bit),
so this information wil be incorrect for very big drives,
such as CD-ROMs (most programs, including Windows File Manager,
have the same problem). If the number of clusters is &Hffff and
the drive type is CDROM, SysInfo/SysInfoNum will now return
650000 for type (DISK_SIZE_KB [+ drive]).

Example SysInfoNum

```
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

```
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

```
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

VBIT Spreadsheet Routines

- ITabCopyDataToVTSS — Dump/write the contents of a table to a Visual Tools spreadsheet.
- ITabCopyFromVTSS — Read the contents of a Visual-Tools spreadsheet to a new table.
- ITabCopyToVTSS — Dump/write the contents of a table to a Visual Tools spreadsheet.
- ITabSetMaxDecimalFromVTSS — Will force the next call to ITabCopyFromVTSS to round decimal numbers to the given number of decimals.
- ITabToVTSS — Export table to a Visual Tools Spreadsheet.
- VTSSget — Returns the contents of a given cell in a Formula One SpreadSheet as string
- VTSSput — Write a string to a given cell in a Formula One SpreadSheet.

Function ITabCopyDataToVTSS

Back

Dump/write the contents of a table to a Visual Tools spreadsheet.
The layout of the spreadsheet (column headers and column width) will not be altered by this call.

Usage:

```
ITabCopyDataToVTSS Handle&, SShandle&
```

This function makes it possible to write Excel 4.0 spreadsheet and *.vts files, internal format of Visual Tools, indirectly via the spreadsheet.

Function ITabCopyDataFromVTSS

Back

Back

Back

Read the contents of a Visual-Tools spreadsheet to a new table. This function makes it possible to read Excel 4.0 spreadsheet and *.vts files, internal format of Visual Tools, indirectly via the worksheet.

The number of lines in the table will be as many as it are datafilled lines in the worksheet. Blank trailing lines in the worksheet are disregarded.

Usage:

```
Handle& = ITabCopyFromVTSS(SShandle&)
```

Remember to delete the table, using ITabDelete, when it is no longer needed (free system resources):

```
MyTab& = ITabCopyFromVTSS(Sheet1.SS)
```

See Also

ITabCopyToVTSS

Sub ITabCopyToVTSS

Back

Back

Back

Dump/write the contents of a table to a Visual Tools spreadsheet.

The contents of line 0 in the table will be used as column headers, and the width of all columns in the spreadsheet will be adjusted to fit the longest string in each column.

Usage:

```
ITabCopyToVTSS Handle&, SShandle&
```

This function makes it possible to write Excel 4.0 spreadsheet and *.vts files, internal format of Visual Tools, indirectly via the spreadsheet.

Display list of all files in the WINDOWS-directory:

```
Windir = SysInfo(DIR_WINDOWS) & "\*.*"
TestTab& = ITabDir(Windir, 6)
ITabCopyToVTSS TestTab&, Sheet1.SS
```

See Also

ITabCopyFromVTSS

Function ITabSetMaxDeciamlFromVTSS

Back

Will force the next call to ITabCopyFromVTSS to round decimal numbers to the given number of decimals.

Usage:

```
ITabSetMaxDecimalsFromVTSS maxdec%
```

NOTE: This is a potentially dangerous call since it will set a global variable in the DLL. The danger is minimized since this variable will be reset by the next call to ITabCopyFromVTSS. Still there is a very small possibility of another application calling ITabCopyFromVTSS before your application, and both applications may get unexpected results.
The chances for this to happen are microscopic, but you should minimize the risk by calling ITabCopyFromVTSS immediately after this call.

Function ITabToVTSS

Back

Export table to a Visual Tools Spreadsheet.

Usage:

```
ITabToVTSS table&, ssHandle&, types&
```

Types:

```
IT_NUM_CONV     ' Number as numeric (not as string)
IT_AUTO_SIZE    ' Automatic justify columnwidth
IT_COL_HEAD     ' Set column heading from line 0
```

Example:

```
' No formatting, no column heading from line 0:
   ITabToVTSS table&, ssHandle&, 0

' As ITabCopyDataToVTSS:
   ITabToVTSS table&, ssHandle&, IT_AUTO_SIZE + IT_NUM_CONV

' As ITabCopyToVTSS, but without rightjustify of numbers
  ITabToVTSS table&, ssHandle&, IT_AUTO_SIZE + IT_COL_HEAD
```

Function VTSSget

Back

Back

Back

Much simplified replacement for the Formula One call SSGetTextRC.
The function returns the contents of a given cell in a Formula One SpreadSheet as string (max 255 char). This call can also be used for reading column headers (row%=0), row headers (col%=0) and the top left cell (0,0).

Usage:

```
result$ = VTSSget(ss&, row%, col%)
```

Read the contents of line 4, column 1 in spreadsheet Sheet1:

```
contents$ = VTSSget(Sheet1.SS,4,1)
```

In Formula One :

```
contents$ = String$(255,0)
err% = SSGetTextRC(Sheet1.SS,4,1,contents$,255)
```

See Also

VTSSput

Function VTSSput

Back

Back

Back

Simplified replacement for the Formula One function SSSetTextRC. VTSSput does not return status, and can also be used for writing column headers (row=0), row headers (col=0) and the top left cell (0,0).

Usage:

```
VTSSput ss&, row%, col%, dataStr$
```

Write data to line 4 - column 1 in spreadsheet Sheet1

```
VTSSput Sheet1.SS, 4, 1, "Test"
```

In Formula One:

```
err% = SSSetTextRC(Sheet1.SS,4,1,"Test")
```

See Also

VTSSget

VBIT Table Routines

ITabBlankLine	Remove contents of a given line in a table
ITabBlankLines	Remove contents of the given lines in a table
ITabCopy	Copy lines from one table to another
ITabCopyToGrid	Copy contents of a table to a GRID.VBX spreadsheet.
ITabDelete	Delete table and release memory
ITabDir	Create a table containing file and/or directory information.
ITabEnvList	Read environment settings into a new table
ITabEnvString	Read environment settings into a new table
ITabFastSort	Extremely fast sorting of a table
ITabFind	Search data , given mask, by column
ITabFindGE	Search for data, given mask, by column in pre-sorted table.
ITabFromString	Create a one-dimensional tabel from string of words
ITabGet	Get data as string type from table
ITabGetColWidth	Get the width of a given column in a table
ITabGetInt	Get data as integer type from table
ITabGetLine	Get data as string type from an array
ITabGetLong	Get data as long type from table
ITabGetNumColumns	Get the defined number of columns a table consists of
ITabGetNumLines	Get the defined number of lines/rows a table consists of
ITabGetReal	Get data as real type from table
ITabGetSize	Get the total amount of consumed memory for a given table
ITabInsertLine	Insert a line at a given line number in a table
ITabInsertLines	Insert lines from a given line number in a table
ITabNew	Create a new table.
ITabNewArray	Create a new table with only one column.
ITabPut	Write data (string) to a cell in a table
ITabPutInt	Write data (integer) to a cell in a table
ITabPutLine	Write data (string) to a line in a table/array
ITabPutLong	Write data (long) to a cell in a table
ITabPutReal	Write data (real) to a cell in a table
ITabRead	Read a table to memory from a file in a specific format
ITabReadFixedRecLenFile	Read a file with fixed record length to a table
ITabReadFmt	Read a file with fixed record length to a new table
ITabRemoveLine	Remove a line at a given line number in a table
ITabRemoveLines	Remove lines from a given line number in a table
ITabSmartSort	Sort a table by column using the SmartSort algorithm
ITabWrite	Write a table to a named disk file of a specific format

Sub ITabBlankLine

Back

Back

Erase the content of the given line.

Usage:

```
ITabBlankLine Handle&, atLine%
```

The line will still be there, but all columns will be empty.

See Also

ITabBlankLines

Sub ITabBlankLines

Back

Back

Erase the content of the specified lines.

Usage:

```
ITabBlankLines Handle&, atLine%, numLines%
```

The lines will still be there, but all columns will be empty.

See Also

ITabBlankLine

Function ITabCopy

Back

Copy lines from one table to another. The destination table will be expanded if needed.
If the source table contains more columns than the destination table, the surplus columns will be ignored.

Usage:

```
ITabCopy  fromTab&, fromLine%, toTab&, toLine%, numLines%
```

Function ItabCopyToGrid

Back

Back

Copy contents of a table to a GRID.VBX spreadsheet.

The contents of line 0 in the table will be used as column headers, and the width of all columns in the spreadsheet will be adjusted to fit the longest string in each column.

Usage:

```
ITabCopyToGRID Handle&, Grid.hWnd
```

Example ITabCopyToGrid

Display list of all files in the WINDOWS-directory:

```
Windir = SysInfo(DIR_WINDOWS) & "*.*"
TestTab& = ITabDir(Windir, 6)
ITabCopyToGRID TestTab&, Grid1.hWnd
```

Sub ITabDelete

Back

Back

Delete a table from memory with effective memory release.

Usage:

```
ITabDelete Handle&
```

NB!

After a table is deleted the handle is invalid. Using a handle for a deleted table will cause an error. Of course, the variable holding the handle value may be reused for new tables.

Example ITabDelete

```
Function FileLength& (ByVal FileName$)
    tempTab& = ITabDir(FileName$, 3)           'create a table using ITabDir
                                               'consists of ? rows and 3 cols
    FileLength& = ITabGetLong(tempTab&, 1, 3) 'Read data from row 1 and col 3
                                               'and return filelength as Long
Integer.
    ITabDelete tempTab&                        'delete table free resources
End Function
```

Sub ITabDir

Create a table with a list of filenames and/or directory names, and optionally, more detailed information connected to these. The function returns a handle to the new table.

Usage:

```
Handle& = ITabDir(FileMask$, Type%)
```

FileMask$ can be a file / directory name, or a standard wildcard mask using the characters: "?" and "*"

```
                Col 1      Col 2  Col 3  Col 4  Col 5  Col 6
Type%
      1      | File.Ext |
      2      | Filename | Ext |
      3      | Filename | Ext | Size |
      4      | Filename | Ext | Size | Date |
      5      | Filename | Ext | Size | Date | Time |
      6      | Filename | Ext | Size | Date | Time | Attr |

      7      As type 6, but includes hidden and system files in
             addition to normal files.
      8      As type 7, but also includes subdirectories .
      9      As type 6, but includes only subdirectories
      10     As type=9, but the resulting table will not include ".\" (current
dir.) and "..\" (parent dir).
```

Date format: "YYYYMMDD"

Time format: "HH:MM:SS"

Attr format: "ADHRS", the single letter will be found in the given position, (D=2, R=4 ...), when the attribute is active.

A: Archive (set when file is changed - used by back-up systems)
D: Directory name
H: Hidden file
R: Read-Only file
S: System file

The number of columns in the newly created table will be equal to Type% up to 6, and 6 for the rest.

Directory names will always be terminated by the character "\". Obs, beware: The directory name can include the ".EXT", in that case the character "\" will be found in column 2 (for types 8 and 9).

Remember to delete the table, using ITabDelete, when it is no longer needed (free system resources).

Example ITabDir

Make a function for returning the size of a given file.
Will return 0 if the file does not exist (not runtime error as FileLen):

```
Function FileLength& (ByVal FileName$)
    tempTab& = ITabDir(FileName$, 3)
    FileLength& = ITabGetLong(tempTab&, 1, 3)
    ITabDelete tempTab&
End Function
```

See Also

ITabDelete

Function ITabEnvList

Back

Back

Back

Create a new table containing all environment strings defined. The table will have two columns where the first column contains the variable name, and the second the environment setting.

Usage:

```
Handle& = ITabEnvList()
```

Remember to delete the table, using ITabDelete, when it is no longer needed (free system resources)

Sample collection

Example ITabEnvList

```
envTab& = ITabEnvList()

' the table may look like this:
'
     1st column   2nd column
     CONFIG       QEMM
     COMSPEC      C:\DOS\COMMAND.COM
     SHARE        ON
     TMP          E:\TMP
     APPEND       D:\PROG
     LIST         E:\TMP
     BLASTER      A220 I7 D1 H5 P330 T6
     LIB          C:\MSVC\LIB;C:\MSVC\MFC\LIB;..\LIB
     WINDIR       D:\WINDOWS

row% = ITabFind(envTab&, "COMSPEC", 1, 1, IT_EXACT)   ' search If row% Then
   CommandPath$ = ITabGet(envTab&,row%,2)       ' read col 2
Else
   CommandPath$ = ""    ' not found
EndIf
ITabDelete envTab&      ' clean up
```

See Also

ITabEnvString

Function ITabEnvString

Back

Back

Back

This function will create a table from a given enviroment variable. The Table will consist of one row pr. "word" which is separated with ;

ITabEnvString is very useful in decoding variables such as **PATH, INCLUDE, LIB** etc

Usage:

```
handle& = ITabEnvString(envVar$)
```

If the variable does not exist, the function rerturns 0 (zero, no table is created)

Remember to delete the table when its no longer needed.

Example ITabEnvString

```
table& =  ITabEnvString("PATH")

the table may then look like this

C:\DOS
c:\windows
d:\bat
e:\PROG
C:\UTIL
.....o.s.v.
```

See Also

ITabEnvList

Function ITabFastSort

Back

Extremely fast sorting of a table. This routine does not format the text in any way during the sorting. The comparison is done directly on the binary data in the table. In other words, the sorting is case sensitive; capital letters are "less than" small letters and non-ascii letters will not likely appear in a logical sorting order. If you need more sofisticated sorting, use ITabSmartSort.

For descending sorting, use negative column.

This routine will not preserve previous sorting order for equality (ITabSmartSort will).

This routine is suitable for preparing a table for the extremely fast binary search routine ITabFindGE.

Usage:

```
ITabFastSort  table&, column%
```

Function ITabFind

Back

Back

Back

Search in a table for data, given column to search in, given from-which-row to search from.

The function returns the row number of the first match-occurrence. If no matching-occurrence is found, the function returns a zero.

Usage:

```
Result% = ITabFind(Handle&, data$, row%, col%, type%)
```

Types:

IT_EXACT	The comparison is done exact.
IT_WILD	Search substring in any position of the column, as a wildcard search "***substring****". The "*" should not be included.
IT_GE	Search substring in position 1 of the column, as a wildcard search "**substring***". The "*" should not be included. Will return the line number of the first line where the corresponding data is greater or equal (GE) than the given substring.

The IT_* parameters are defined as global constants in then file "VBIT.BAS"

Example ITabFind

```
FileSubstStr "MYPROG.TXT", "OLDLIB", "NEW LIB"
```

This call is supposed to open the file "MYPROG.TXT", replace all occurences
of the string "OLDLIB" with "NEW LIB" and write the file back to disk.
The code for this task can be written like this:

```
Sub FileSubstStr (ByVal FileName$, ByVal FromStr$, ByVal ToStr$)
    table& = ITabRead(FileName$, IT_TEXTFILE)
    row% = 0
    Do
      row% = ITabFind(table&, FromStr$, row% + 1, 1, IT_WILD)
      If row% = 0 Then Exit Do
      ITabPutLine table&,row%,SubstAll(FromStr$,ToStr$,ITabGetLine(table&,row
%))
    Loop
    ok% = ITabWrite(table&, FileName$, IT_TEXTFILE)
    ITabDelete table&
End Sub
```

The line ITabPutLine
table&,row%,SubstAll(FromStr$,ToStr$,ITabGetLine(table&,row%))
may look unreadable, but this illustrates the power of routines returning strings that can be used directly as an argument to another routine and so on.

The line could have been split into 3 lines like this:

```
temp1$ = ITabGetLine(table&, row%)
temp2$ = SubstAll(FromStr$, ToStr$, temp1$)
ITabPutLine table&, row%, temp2$
```

See Also

ITabFindGE

Function ITabFindGE

Back

Back

Back

Search in a sorted table, given column, for "data*". The data comparison is exact. Folded/not folded letters are evaluated differently. It is essential that the table is pre-sorted. The function returns the row number of the first match-occurrence which is greater or equal(GE). If no matching-occurrence is found, the function returns a zero.

Usage:

```
Result% = ITabFindGE(Handle&, findStr$, col%)
```

Example ITabFindGE

A very fast way to look up data from a huge ascii file can be done this way:
An ascii file consists of 20,000 lines where each line is 80 + 2 positions long. (Cr/LF=2). In a VB loop the 8 first characters of each line in the ascii file is read into an array. The file is assumed to be sorted.

To get hold of data from the ascii file:
Search in the table and get match based on the 8 characters. If match, the function ITabfindGE returns the row number. Knowing the
fact that each line is 82 bytes long, the exact bytes position within the ascii file is [(matching row number-1) * 82].

```
Dim Found, BytePos As integer
Dim DataLine As string
Found = ITabFindGE(MyTable&, "1234PROD", 1)      'Search in the table
BytePos = (Found-1) *  82                        'Knowing the  absolute byte

                                                 'position the Basic operators
are
                                                 'used:
DataLine=String(82," ")                          'Define the variable to be
read in
Open  "Data.Txt"  #1                             'Open the ascii file
Seek  #1, , BytePos                              'Set file pointer to exact
position in the file
Get   #1, , DataLine                             'Get the data line from the
file
Close #1                                         'Close the ascii file
```

See Also

ITabFind

Function ITabFromString

Back

Create a one-dimensional tabel from string of words separated with a given delimiter character.

Usage:
Handle& = ITabFromString(streng$, skilletegn$)

Words that are enclosed with the Chr$(34) may contain the delimiter character.

Example 1)
handle& = ItabFromString("word1;word2;word3",";")

we would create a tabel of 3 rows (one column):
word1
word2
word3

Example 2)
suppose string$ contains the following :
col1,coloumn 2,"column 3, with delim. char",column4

```
handle& = ItabFromString(string$,",")
```

we would then have created a table looking like this:
col1
column 2
column 3, with delim. char
column4

Example 3)
```
handle& = ItabFromString("D:\WINDOWS\UTIL\BITMAP\PIC.BMP,"\")
```

Will create a table with the following lines.
D:
WINDOWS
UTIL
BITMAP
PIC.BMP

First line line will contain the disk-name, and the last line will contain the filename.

Function ITabGetColWidth

Back

Back

The function returns the width, as an integer, of a given column

Usage:

```
Result% = ITabGetColWidth(Handle&, Col%)
```

See Also

ITabGet
ITabGetInt
ITabGetLine
ITabGetLong
ITabGetNumColumns
ITabGetNumLines
ITabGetReal
ITabGetSize

Function ITabGet

Back

Back

Back

Read data from a cell in a table.

Usage:

```
Result$ = ITabGet(Handle&, Row%, Col%)
```

Example ITabGet

'Display search path in a ListBox:

```
eTab& = ITabEnvList()
row%  = ITabFind(eTab&, "PATH", 1, 1, IT_EXACT)
path$ = ITabGet(eTab&, row%, 2) ' e.g. "C:\DOS;C:\WINDOWS;D:\UTILS;E:\PROG" i
%=1
Do
    p$ = PickWord(path$, i%, Asc(";"))
    If Len(p$) = 0 Then End Loop
    List1.AddItem p$
    i% = i% + 1
Loop
ITabDelete eTab&
```

See Also

ITabGetColWidth
ITabGetInt
ITabGetLine
ITabGetLong
ITabGetNumColumns
ITabGetNumLines
ITabGetReal
ITabGetSize

Function ITabGetInt

Back

Back

Read data from a cell in a table and return an Integer.

Usage:

```
Result% = ITabGetInt(Handle&, Row%, Col%)
```

ITabGetColWidth
ITabGet
ITabGetLine
ITabGetLong
ITabGetNumColumns
ITabGetNumLines
ITabGetReal
ITabGetSize

Function ITabGetLine

Back

Back

Read data from a line in a table. This is practical when reading rows from tables with only one column.

Usage:

```
Result$ = ITabGetLine(Handle&, Row%)
```

See also

ITabGetColWidth
ITabGet
ITabGetInt
ITabGetLong
ITabGetNumColumns
ITabGetNumLines
ITabGetReal
ITabGetSize

Function ITabGetLong

Back

Back

Read data from a cell in a table and return a Long Integer.

Usage:

```
Result& = ITabGetLong(Handle&, Row%, Col%)
```

See also

ITabGetColWidth
ITabGet
ITabGetInt
ITabGetLine
ITabGetNumColumns
ITabGetNumLines
ITabGetReal
ITabGetSize

Function ITabGetNumColumns

The function returns the number of columns the table consists of.

Usage:

```
Result% = ITabGetNumColumns(Handle&)
```

Example ITabGetNumColumns

Find out how many columns there are in a TAB-delimited file:

```
aTab& = ITabRead("DATAFILE.CSV", IT_CSVFILE + 9)
numCols% = ITabGetNumColumns(aTab&)
ITabDelete(aTab&)
```

See also

ITabGetColWidth
ITabGet
ITabGetInt
ITabGetLine
ITabGetLong
ITabGetNumLines
ITabGetReal
ITabGetSize

Function ITabGetNumLines

This function returns the current number of lines the table.

Usage:

```
Result% = ITabGetNumLines(Handle&)
```

Note: The number of lines in a table is not static.
Several routines are capable of changing the number of lines i a table:

ITabInsertLine
ITabInsertLines
ITabRemoveLine
ITabRemoveLines

Calling ***ITabGetNumLines*** is normaly the logical thing to do after the following calls:

ITabRead
ITabReadFixedRecLenFile
ITabEnvList
ITabDir
ITabCopyFromVTSS

Example ITabGetNumLines

```
table& = ITabRead ("\AUTOEXEC.BAT", IT_TEXTFILE)

lines% = ITabGetNumLines(table&)
```

Note: is lines%=0, there are two possibilities:
A) the file exists and has 0 lines
B) the file does not exist (suppose we had the wrong drive?)
If the difference is significant, you should check for the existence
of the file before you attempt to read it.

```
' The following function determines whether a file exist or not:

Function FileExist% (ByVal FileName$)
    tempTab& = ITabDir(FileName$, 1)
    If ITabGetNumLines(tempTab&) Then
       FileExist% = True
    Else
       FileExist% = False
    End If
    ITabDelete tempTab&
End Function


Please refer to FileExist function.
```

See also

ITabGetColWidth
ITabGet
ITabGetInt
ITabGetLine
ITabGetLong
ITabGetNumColumns
ITabGetReal
ITabGetSize

Function ITabGetReal

Back

Back

Read data from a cell in a table and return a Real (double precision floating point) number.

Usage:

```
Result#  =  ITabGetReal (Handle&, Row%, Col%)
```

See also

ITabGetColWidth
ITabGet
ITabGetInt
ITabGetLine
ITabGetLong
ITabGetNumColumns
ITabGetNumLines
ITabGetSize

Function ITabGetSize

Back

Back

The function returns the size of a given table in bytes. Once the table is dimensioned by the ITabNew operator, the table does not occupy memory space of any consideration. When data is loaded into the table an, increase in memory consumption can be
observed. The memory consumption is dynamic and depends on the amount of loaded data.

Usage:

```
Result& = ITabGetSize(Handle&)
```

See also

ITabGetColWidth
ITabGet
ITabGetInt
ITabGetLine
ITabGetLong
ITabGetNumColumns
ITabGetNumLines
ITabGetReal

Sub ITabInsertLine

Back

Back

Insert a blank line/row in a table at a given line number, atLine%. The new inserted line will contain blank cells in all columns. The lines at and below the insert point will be pushed down one position. The number of lines in the table will be affected/changed, see ITabGetNumLines(+1).

Usage:

```
ITabInsertLine Handle&, atLine%
```

See also

ITabInsertLines

Sub ITabInsertLines

Back

Back

Insert blank lines/rows in a table at a given line number, atLine%. The new inserted lines will contain blank cells in all columns. The lines at and below the insert point will be pushed down as many positions as the number of inserted lines, numLines%. The number of lines in the table will be affected/changed, see ITabGetNumLines(+n)

Usage:

```
ITabInsertLines Handle&, atLine%, numLines%
```

See also

ITabInsertLine

Function ITabNew

Back

Back

Back

Create and dimension a new table. Returns a handle which will identify the table.
May be used to create table with 0 rows.

Usage:

```
Handle& = ITabNew(rows%, columns%)
```

Remember to delete the table, using ITabDelete, when it is no longer needed (free system resources).

Sample collection

Example ITabNew

```
'Define a table consisting of 100 rows and 10 columns pr row.
Mytab&= ITabNew(100, 10)
```

See also

ITabNewArray

Function ITabNewArray

Back

Back

Back

Create a table with one column . The function returns a handle which will identify the table.
May be used to create table with 0 rows.

Usage:

```
Handle& = ITabNewArray(ByVal lines%)
```

Remember to delete the table, using ITabDelete, when it is no longer needed (free system resources).

Example ITabNewArray

```
Define a table consisting of 100 lines (one column).
Mytab&= ITabNewArray(100)
```

See also

ITabNew

Sub ITabPut

Back

Back

Put string data into a cell in the table.

Usage:

```
ITabPut Handle&, Row%, Col%, DataString$
```

See also

ITabPutInt
ITabPutLine
ITabPutLong
ITabPutReal

Sub ITabPutInt

Back

Back

Put numeric (integer) data into a cell in the table.

Usage:

```
ITabPutInt  Handle&, Row%, Col%, IntegerNumber%
```

See also

ITabPut
ITabPutLine
ITabPutLong
ITabPutReal

Sub ITabPutLine

Back

Back

Put data into a line in the table (as string). This is a practical call for writing lines to tables with only one column.

Usage:

```
ITabPutLine Handle&, Row%, DataString$
```

See also

ITabPut
ITabPutInt
ITabPutLong
ITabPutReal

Sub ITabPutLong

Back

Back

Put numeric data into a cell in the table (as Long).

Usage:

```
ITabPutLong Handle&, Row%, Col%, LongNumber&
```

See also

ITabPut
ITabPutInt
ITabPutLine
ITabPutReal

Sub ITabPutReal

Back

Back

Put numeric data into a cell in the table (as Double).

Usage:

```
ITabPutReal Handle&, Row%, Col%, DoubleRealNumber#
```

See also

ITabPut
ITabPutInt
ITabPutLine
ITabPutLong

Function ITabRead

Back

Back

Read a file into a new table. The table is dimensioned depending on the contents of the disk file.

Usage:

```
Handle& = ITabRead(FileName$, FileType%)
```

Remember to delete the table, using ITabDelete, when it is no longer needed (free system resources):

Filetype:

IT_TABFILE	Read an earlier written table.
IT_TEXTFILE	An ordinary text file
IT_CSVFILE + Delim.	Read a file where the columns are delimited by a given character
IT_CSV0FILE + Delim	As above, but the first line in the file is written to line zero in the table (typically for column headers)

The **IT_*** parameters are defined as global constants in the file "**VBIT.BAS**"

Additional parameters:

+ IT_ASCII	Translate from DOS characters to Windows characters
+ STRIP_T	Remove trailing blanks

Delim is the ascii value of then delimitter character:Tab=9,(Asc";"),(Asc",")

If a table file, IT_TABFILE, is read, the file that was written from a table as source, the table will gain the same dimension as the source table had.

If an ordinary textfile, IT_TEXTFILE, is read, the table will become an array.

Remember to delete the table, using ITabDelete, when it is no longer needed (free system resources):

When reading a CSV-file, the character Chr$(34), will be taken into consideration.
Columns that are enclosed with Chr$(34), may contain the delimiter character. Chr$(34) will be removed during read.

Consider this :
col1,column 2,"column 3, with Delim char ,",column4

woud be read like this :

col1
column 2
column 3, with Delim char ,
column4

Example ITabRead

Read an earlier written table file named "Written.Tab" to memory. How the table is dimensioned is determined by the dimension of the read table/file "Written.Tab"

```
Mytab&= ITabRead("Written.Tab", IT_TABFILE)
```

Read a textfile named "Text.fil" to memory and pr line remove trailing blanks, optional, and translate from DOS to Windows characters-set, optional. The table becomes an array.

```
Mytab& = ITabRead(Text.fil,IT_TEXTFILE[+ STRIP_T][ + IT_ASCII])
```

Function ITabReadFixedRecLenFile

Back

Back

Read a file with fixed record length to a new table. The function returns the table handle.

Usage:

```
Handle& = ITabReadFixedRecLenFile(FileName$, fmt$)
```

Remember to delete the table, using ITabDelete, when it is no longer needed (free system resources):

A linefeed between each record is assumed. The "fmt$" parameter tells the system what to be picked from the record and placed to which column in the table. Capitalised letters (A-Z) are used for giving the position and length (repeated) in the record, The letter used. also indicates which column the data is to be put into. "A" is column 1, "B" = 2 and "C" = 3..."Y" = 25 and "Z" = 26. The width is given by repeating the letter. "A" means, pick one character and put it in column 1. "BBBB" means pick four characters and put it in column 2.
"ZZZ" means 3 characters to column 26. The sequence can mixed, order, according to the datafile, as long as "A" comes before "B".
Further on empty columns can be reserved in the table by skipping letters in the sequence. Leading and trailing blanks will be stripped before data is put to the table.

From version 1.24, it is possible to read more than 26 columns using characters following "Z" in the ascii table: "...XYZ[\]^_`abcde..."
up to ascii value 126 ("~"). This makes it possible to read up to
62 columns.

Example ITabReadFixedRecLenFile

The first line shows the "fmt$" and the 10 to follow the datalines within a datafile:

```
"AAAAAAAAAAAAABBBBBBBB  DD CC EE  FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"
 File Name    Size      Date      D e s c r i p t i o n
 ========================================================================
 TOLL20.ZIP   35652     04-28-93  Tool Button Custom Control For VBasic
 VB2_TB.ZIP   162895    01-14-93  The Ultimate VBASIC v2.0 Add-On, Supe
 VB4EX.ZIP    116486    03-14-92  Example Of How To Use DLL's With VisB
 VBE1NG.ZIP   118521    04-04-93  Visual Basic Engine For Making DataBa
 VGX3.ZIP     194111    01-06-93  VGA Graphics File Lib For QB/BASIC Pr

 VXBASE.ZIP   212606    03-25-92  XBase Windows Visual BASIC Functions
 VXBDOC.ZIP   132274    03-19-92  XBase Windows Visual BASIC Docs [2/2]
 WBB12.ZIP    266520    09-10-92  BasicBasic For Windows v1.2

The table defined with 6 columns and 10 lines:
1.           2.        3. 4. 5.  6.column:
File Name    Size      e  Da     D e s c r i p t i o n
============ ========  == == ==  ====================================
TOLL20.ZIP   35652     28 04 93  Tool Button Custom Control For VBasic
VB2_TB.ZIP   162895    14 01 93  The Ultimate VBASIC v2.0 Add-On, Supe
e.t.c.
```

Function ITabReadFmt

Back

Read a file with fixed record length to a new table. The function returns the table handle.

Usage:

```
Handle& = ITabReadFmt& (fileName$, fmt$, type%)
```

fmt$: see ITabReadFixedRecLenFile

type%:

IT_STRIP_L		: Remove leading (left)
IT_STRIP_T	or IT_STRIP_R	: Remove trailing (right)
IT_STRIP_LT	or IT_STRIP_LR	: Both leading and trailing
IT_STRIP_ALL		: Remove all spaces

Function ITabRemoveLine

Back

Back

Remove a line/row in a table from a given line number, atLine%. The lines below the given linenumber will be scrolled up one line.
The number of lines in the table will be affected/changed, ITabRemoveLines (-1)

Usage:

```
ITabRemoveLine Handle&, atLine%
```

See also

ITabRemoveLines

Function ITabRemoveLines

Back

Back

Remove lines/rows in a table from a given line number, atLine%. The lines below the given linenumber, atLine%, + the number of deleted lines, num%, will scroll up. The number of lines in the table will be affected/changed, ITabGetNumLines (-n)

Usage:

```
ITabRemoveLines Handle&, atLine%, num%
```

See also

ITabRemoveLine

Sub ITabSmartSort

Back

Back

Sort a table by the contents of the given column. This rutine sorts both text and numbers logically. The sorting is not case sensitive. If there are duplicates in the column beeing sorted, the original order will be kept. This makes it possible to sort on several columns just by repeating this call (sort least significant column first).

Usage:

```
ItabSmartSort Handle&, Col%
```

If the column number is negative, the sorting will be descending on column (-Col%).

Example ITabSmartSort

```
Given then table, "TestTab", consisting of one column and data as follows:
      Number 1 of 100
      Number 10 of 100
      Number 100 of 100
      Number 2 of 100
      Number 20 of 100
      Number 20 of 50
(The result of an ordinary sort in Excel)
```

Call ITabSmartsort(TestTable&, 1)

The result of "smart"sorting :

```
      Number 1 of 100
      Number 2 of 100
      Number 10 of 100
      Number 20 of 50
      Number 20 of 100
      Number 100 of 100
```

Function ITabWrite

Write a table to a disk file. Returns error code (always 0 in this version).

Usage:

```
Result% = ITabWrite(Handle&, FileName$, FileType%)
```

FileType:

IT_TABFILE	The internal format for reading/writing
IT_TEXTFILE	Ordinary textfile which can be read to a table consisting of one column

Additional options:

IT_ASCII	Translate from Windows to the DOS character set.

The IT_* parameters are defined as global constants in VBIT.BAS.

Revision History

Ver 1.39

New function ITabToVTSS

Ver 1.33

Bug fix for FormNum (decimals=0)

Ver 1.32

Added : Function FormNum

Ver 1.31

Added : Function ITabReadFmt

Ver 1.30

ItabNew, ItabNewArray is expanded. May now be used to create new tables with 0 lines/rows.

Ver 1.29

BUG Fix

The function "Place$(from$,to$,pos%,num%)" caused wrong result (and sometimes GP error) when length of "from$" was bigger than "num%". Corrected.

The following routines should now work for dates from 1/1 1800 (day 1) to 28/2 2400

```
dayNumber&    = GetDayNumber(dateStr$, dateFmt$)
dateAsLong& = GetDateLong(dayNumber&)
dateString$ = GetDateStr(dateNum&, dateFmt$)
```

Ver 1.28

New functions

GetDateLong
GetDateStr
GetDayNumber

Ver 1.26

Changed the following routines:

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

Before : ' Dates must be between year 1901 and 2199
Now : ' Valid results for dates from September 14th 1752
' to December 31 9999.

Bug-fix in

SysInfo(DIR_WINDOWS)
SysInfo(DIR_WINDOWS_SYSTEM)

If path was the root, these routines would return "P:\\".
All routines in VBIT returning path are supposed to return one (and only one) trailing "\" regardless of the path beeing root or not. The reason for this is that the application programmer should not have to mess up the application program with tests for root and appending "\", as you have to when

using the standard Visual Basic routines and API calls for this.

Ver 1.25

NEW routines:
GetNumDays&
Interest#

Changed routine
Sound

Ver 1.24

Expanded ITabReadFixedRecLenFile:

From version 1.24, it is possible to read more than 26 columns using characters following "Z" in the ascii table: "...XYZ[\]^_`abcde..." up to ascii value 126 ("~"). This makes it possible to read up to 62 columns.

NEW routine:
SOUND

Ver 1.21

Bug-fix:
ITabDir with type=9 went into an endless loop ("hang") when number of matching subdirectories was 0.
This problem would typically occur on a diskette.

New features:
+ ITabDir - new type=10:
As type=9, but the resulting table will not include ".\" (current dir.) and "..\" (parent dir).

+ SysInfo:
DISK_TYPE can now return "CDROM" (returned "REMOTE" before)

+ SysInfo and SysInfoNum - new types:
+ DISK_CLUSTER_SIZE [+ drive]
+ DISK_SECTOR_SIZE [+ drive]
+ DISK_SECTORS_PR_CLUSTER [+ drive]
+ DISK_AVAILABLE_CLUSTERS [+ drive]
+ DISK_TOTAL_CLUSTERS [+ drive]

Note: Number of clusters is limited to &Hffff (16 bit),
so this information wil be incorrect for very big drives,
such as CD-ROMs (most programs, including Windows File Manager,
have the same problem). If the number of clusters is &Hffff and
the drive type is CDROM, SysInfo/SysInfoNum will now return
650000 for type (DISK_SIZE_KB [+ drive]).

Ver 1.20

Official release April 10th 1995

Ver 1.19

Errors in ITabFastSort and ITabFindGE corrected.

There was a possiblity for GPF if the tabel contained blank lines.

ITabFastSort and ITabSmartSort will show hourglass while sorting is in progress.

Strip function changed. Second parameter for function should now be a string.

Ver 1.18

NEW table functions
- IT_GE parameter type added to ITabFind function
- Correct BROWS sequence in HLP file, minor corrections

Ver 1.17

NEW table functions
- ITabCopy
- ITabFastSort

Ver 1.16

NEW file functions
- FileExist
- FileGetAttr
- FileGetDate
- FileGetExt
- FileFindPath
- FileGetFileName
- FileGetPath
- FileGetSize
- FileGetTime

Ver 1.15

Name changed
- Several function are renamed !
- Please read Important infomation

Ver 1.14

NEW functions Spreadsheet
- VTSSget
- VTSSput
- ITabSetMaxDecimalsFromVTSS

NEW functions Table
- ITabEnvString
- ITabFromString

Ver 1.13

+ ITabCopyToVTSS will not clear cell formatting any

more (i.e. allignment, formatting of numbers etc).

+ ITabCopyToVTSS will not cause blank cells to be displayed as 0 any more.

+ New function: ITabCopyDataToVTSS.
Similar to ITabCopyToVTSS, but will not adjust column width and not use
row 0 as column headers.

+ IDBTOOLS.HLP added to the package.

Ver 1.12

+ ITabRead* did not handle files with more than 16383
lines consistantly. The number of lines read was
determined by the actual number of lines & &H3fff...

Now it will read up to 16128 lines. If reported
number of lines is bigger than 16000, you can
assume that not all of the file has been read.

+ ITabCopyToVTSS will not cause blank cells to be displayed as 0 any more.

+ New function: ITabCopyDataToVTSS.
Similar to ITabCopyToVTSS, but will not adjust column width and not use
row 0 as column headers.

+ IDBTOOLS.HLP added to the package.

+ New call: ITabSetMaxDecimalsFromVTSS(maxDec%)
Will force the next call to ITabCopyFromVTSS to
round decimal numbers to maxDec% decimals.
NOTE: This is a potentially dangerous call since
it will set a global variable in the DLL. The danger
is minimized since this variable will be reset by the

next call to ITabCopyFromVTSS. Still there is a very
small possibility of another application calling
ITabCopyFromVTSS before your application, and both
applications may get unexpected results. The chances
for this to happen are microscopic, but you should
minimize the risk by calling ITabCopyFromVTSS
immediately after this call.

Ver 1.11

+ Bug fix in ITabPutReal: got protection error when
writing small negative numbers with many decimals.
This call will no longer convert very big or very
small numbers to scientific notation (eñnnn), but
instead store all digits (may cause long strings
for very big/small numbers). This should not be a
problem for real life applications....

+ ITabCopyFromVTSS will now convert decimal numbers to
strings using "." (period) as the decimal delimiter,

regardless of the system settings. This makes it

possible to read such numbers as strings into variants getting the expected result.

Sample Collection

- AnsiToAscii Sample
  Sample using the AnsiToAscii function to convert from Windows to DOS character set. Also showing the special character sets for Norway and Denmark
- DosToUnix Sample
  Function converting from DOS to UNIX character set
- UnixToDos Sample
  Converting from UNIX to DOS character set
- FileSubstStr Sample
- GetNumWords Sample
  Return number of words in a string given a delimiter
  Leading, trailing and repeted embedded delimiters are ignored
- MakeArray Sample
  Sample showing the difference between ITabNew and VB array
- ShowPath Sample
  Display search path in List1

```
Sub AnsiToAsciiSample ()
    Open "scan-dos.txt" For Output As #1
    Print #1, "In Norway and Denmark, we use some special characters:"
    Print #1, AnsiToAscii("[Æ]=[AE], [Ø]=[OE] and [Å]=[AA]")
    Print #1, AnsiToAscii("[æ]=[ae], [ø]=[oe] and [å]=[aa]")
    Print #1, AnsiToAscii("In Sweden, they use [Ä] instead of  [Æ],")
    Print #1, AnsiToAscii("[ä]=[æ], [Ö]=[Ø] and [ö]=[ø].")
    Close #1
End Sub
```

```
Sub DosToUnix (ByVal FromFile$, ByVal ToFile$)
    BytesToRead& = FileLen(FromFile$)
    If FileLength(ToFile$) > 0 Then Kill (ToFile$)
    Open FromFile$ 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$(BuffSize&, #1), 10) ' Read and convert LF
to CR/LF
           ' NB: Problem if CR/LF is found exactly at a maxBuff& boundary:
       If Asc(Pick(buffer$, BuffSize&, 1)) = 13 Then ' Fix it:
           buffer$ = Pick(buffer$, 1, BuffSize& - 1)  ' remove CR (last chr)
       End If
       Put #2, , buffer$
       BytesToRead& = BytesToRead& - BuffSize&
    Loop
    Close #1
    Close #2
End Sub
```

```
Sub UnixToDos (ByVal FromFile$, ByVal ToFile$)
    BytesToRead& = FileLen(FromFile$)
  ' If FileLength(ToFile$) > 0 Then Kill (ToFile$)
    If FileExist(ToFile$) Then Kill (ToFile$)
    Open FromFile$ 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$(BuffSize&, #1), -10) ' Read and convert CR/LF to
LF
       Put #2, , buffer$
       BytesToRead& = BytesToRead& - BuffSize&
    Loop
    Close #1
    Close #2
End Sub
```

```
Sub FileSubstStr (ByVal FileName$, ByVal FromStr$, ByVal ToStr$)
    table& = ITabRead(FileName$, IT_TEXTFILE)
    row% = 0
    Do
       row% = ITabFind(table&, FromStr$, row% + 1, 1, IT_WILD)
       If row% = 0 Then Exit Do
       ITabPutLine table&, row%, SubstAll(FromStr$, ToStr$,
ITabGetLine(table&, row%))
    Loop
    ok% = ITabWrite(table&, FileName$, IT_TEXTFILE)
    ITabDelete table&
End Sub
```

```
' Return number of words in a string given a delimiter
' Leading, trailing and repeted embedded delimiters are ignored '
Function GetNumWords% (ByVal FileMask$, ByVal Delim$)
    bs% = -Asc(Delim$)
    n% = 0
       While Len(PickWord(FileMask$, n% + 1, bs%))
            n% = n% + 1
       Wend
    GetNumWords% = n%
End Function
```

```
Sub MakeArray ()
    Const x% = 5000
    Const y% = 10
 '   Static BigArr(x%, y%) As String
 '   For i% = 1 To x%
 '       For j% = 1 To y%
 '           BigArr(i%, j%) = "TESTING"
 '       Next j%
 '   Next i%

    BigTable& = ITabNew(x%, 10)
      For i% = 1 To x%
            For j% = 1 To 10
                  ITabPut BigTable&, i%, j%, "TESTING " & i% & "," & j%
            Next j%
      Next i%

  '  XXX_TEST!List1.Visible = True
  '  XXX_TEST!List1.ZOrder
    For i% = (x% - 100) To x%
      For j% = 1 To y%
    '        XXX_TEST!List1.AddItem BigArr(i%, j%)
    '        XXX_TEST!List1.AddItem ITabGet(BigTable&, i%, j%)
      Next j%
    Next i%

End Sub
```

```
Sub ShowPath ()
' Display search path in List1:
 eTab& = ITabEnvList()
 row% = ITabFind(eTab&, "PATH", 1, 1, IT_EXACT)
 path$ = ITabGet(eTab&, row%, 2) ' e.g. "C:\DOS;C:\WINDOWS;D:\UTILS;E:\PROG"
 i% = 1
 Do
    p$ = PickWord(path$, i%, Asc(";"))
    If Len(p$) = 0 Then Exit Do
 '>>  List1.AddItem p$
    i% = i% + 1
 Loop
 ITabDelete eTab&
End Sub
```

Ordering Information

LICENSE-CODES / PRICE

Contact Traders Mascot AS to get a license code.
Current prices are:
In Norway: NOK 495 incl. M.V.A.
All other countries: US $ 65
A valid license code gives you the right to distribute VBIT.DLL
with your programs.

ORDERING INFORMATION / PAYMENT

Send to:

Traders Mascot AS
Postboks 3098 Sentrum
N-6001 AALESUD, Norway
Phone : +47 7012 1120
Fax : +47 7012 4090
e-mail: bjonor01@mimer.no or infotech@telepost.no

The following information must accompany the payment:

Name _____

Address _____

Postcode _____

City _____

Country _____

Date _____ Phone: _____

Has paid _____ for VBIT license (see PRICE above):
[_] Check / money order enclosed
[_] Visa [_] MasterCard [_]EuroCard
[_] American Express [_] Diners International
[_] BankGiro: 5353.05.22667 (Den Norske Bank)
[_] PostGiro: 0826.06.97588 Post cheque

If American Express, cards serial number :_____

Credit card number :_____

Expiration date :_____

Signature :_____

Want to receive the license code [_] via E-mail: _____
[_] via post.
[_] via Fax: _____

VBIT version:_____

Where did you find VBIT? _____

_____

If you pay directly to the bank account or international Post services,you may prefer to send the above information as E-mail via internet to:
infotech@telepost.no or to bjonor01@mimer.no
We will send the code to you as soon as we have confirmed the payment.

You may find the latest information and releases of this great software package at the following WWW page.

http://www.prodat.no/infotech/

Trader's Mascot AS

Postboks 3098 Sentrum
6001 AALESUND
NORWAY

Phone : +47 7012 1120
Fax : +47 7012 4090
BBS : +47 7012 9014

InterNet e-mail : **bjonor01@mimer.no**

WWW Server : **Coming Soon !**
(www.traders.no)

InfoTech AS
Strandgaten 207
5004 Bergen
NORWAY
Phone : +47 5590 0260

InterNet e-mail : **infotech@telepost.no**

WWW Pages : **http://www.prodat.no/infotech/**