A Visual BASIC Toolbox DLL

http://vbtoolbox.kerys.co.uk

Documentation and Programmer's Guide

Version 1.36

26th March 2011

About the Library

VB is an excellent language to work in but it is lacking in some areas. These deficiencies can be made up for by writing your own functions such as **Round()**, **Min()** or **Max()** in VB to add flexibility. However, for speed-critical routines this may not be ideal or quick enough Interfacing with a DLL gains the "best of both worlds" in terms of high-performance and convenience of use.

The lack of library include features in VB means that creating a pool of regularly used code modules isn't as easy as with languages such as "C" and it can be a pain to have to "re-invent" the wheel to perform common tasks which are not provided-for by the BASIC language itself. This library offers a number of routines, commonly implemented in VB code but which can now be called directly from a DLL without the need for longhand code.

This guide is intended to help programmers make the best use of the features of this extension DLL.

Development and Testing

The DLL functions can be called from any programming language which supports linkage to external DLLs - not just Visual BASIC. For example, Visual FoxPro, VB for Apps (VBA), FreeBASIC etc.

The library interfaces were tested in Microsoft Visual BASIC V5.0 and should work OK with Visual BASIC 6.0 and clones such as PowerBASIC or FreeBASIC.

Although not a commercial package, VBToolbox undergoes periodic testing and revision. Since the released code may contain more recent updates, please ensure that you refer to the current "declares" defined in the accompanying BAS or TLB module where they differ from the contents of this manual.

Test programs are usually included with the distribution and other demonstration programs and resources are available at http://vbtoolbox.kerys.co.uk/

Previous Versions

All previous versions of this DLL should be discarded forthwith from 26 March 2011

General Programmer's Notes On Using the Library

Description

VB is a great language for non-professional programmers, with both VB5 and VB6 still widely used as a commercial development platform. Although the language has broad coverage in terms of functionality there are a few constructs which need regular inclusion in many programming projects such as rapid pointer-handling which can produce really quick code.

Visual BASIC has poor include library support with no automatic include-chaining as with C/C++. There is an advantage in terms of speed, flexibility and code-reuse in using an external DLL which provides many of the missing features which usually have to be written "longhand". This DLL library can be used not only by VB programmers but also by any language which can safely call external DLLs such as Visual FoxPro. PowerBASIC or FreeBASIC.

This project was originally developed for sole use by the author. However it is shared with the wider computing community in the hope it may be of some use.

Licensing and Terms of Use

The library is offered as royalty-free *freeware* in the hope it may benefit anyone getting to grips with VB for the first time. No warranty of any kind is intended, offered or issued. No liability is accepted for losses of any kind consequential to the use of this product.

The library is expressly intended primarily for educational use by the computer hobbyist. It is not intended for commercial use; nor may it be distributed with commercial software or included in any software library of any kind without express permission from the author. Non-commercial distribution is royalty-free and is permitted as long as these licence conditions are honoured. The software may not be resold or distributed at a profit without the express permission of the author.

You may freely distribute the ANSI or Unicode DLLs with your personal software project but they may not be distributed with any kind of commercial software.

All functionality should be subjected to thorough testing by the programmer before using on any project handling live data. The author accepts no responsibility whatsoever for lost data or damages arising from the use of this software either direct or consequential. By using it you agree to the terms and conditions and agree to indemnify the author against any and all direct or indirect legal liabilities.

Please see any enclosed README.TXT file for the latest licensing conditions and other important information which may affect your licence to use this product.

Unicode and ANSI Versions of this Library

From v1.31 this library is available as either an ANSI or part-Unicode DLL. Full Unicode support is not yet provided and the interface to many string functions still receives ANSI-translated strings from Visual BASIC. See the section relating to each function for more specific information.

The ANSI version consists of a DLL (MSLIB145.DLL) and a set of Declare statements (MSLIB145.BAS) which you can selectively include in your own projects.

Unicode is a special method of handling Strings within programs which uses a 16-bit (wide) character set instead of the traditional ANSI 8-bit (narrow) character set. This permits a much wider range of characters to be represented, including a huge range of international characters.

The Unicode version consists of a DLL (MSLIB146.DLL) and a Type Library (TLB) file (MSLIB146.TLB). You can also use appropriate Declare statements instead of the TLB file but care should be taken when translating from TLB format to Declare format.

Full Unicode support is fairly limited at present and there are few functions which present a need to handle full 16-bit or double-byte international characters.

Declares (ANSI Include File - MSLIB145.BAS)

VB5 Declare statements you will need for projects which use this DLL are included in MSLIB145.BAS.

You will need to add this, either selectively, in part, or in-full, to your project as a VB module. Place a copy of the file in your "include" folder and drag it into the project's "project explorer tree". If you are unsure of the calling-conventions then open this file and examine the function prototypes. Please **do not** change the declares for the downloaded version of the DLL or your VB program will crash or behave unpredictably - particularly where ByRef calls are changed to ByVal. Little or no type-checking is made by VB on variables passed to external DLLs.

Type Library (Unicode Version - MSLIB146.TLB)

To use the Type Library file, from within VB you should select Project->References, browse to the location of the MSLIB146.TLB file and select it using the File Open dialogue. Ensure the module is ticked in the "Available References" list once you have selected it.

Once you have successfully loaded the Type Library Visual BASIC will have the information it requires for you to use the exported DLL functions without requiring Declare statements.

You may browse the exported function list by pressing the F2 key within Visual BASIC, selecting "VBToolbox" from the topmost "<All Libraries>" scrollbox. Help and information about each object is shown at the bottom of the screen.

Functions which process binary data such as **EncryptString** do not handle data as Unicode

One you have compiled your finished program you will no longer require the TLB file.

The type library does not offer full Unicode support. Full support is still in progress.

VBToolbox and Unicode Strings

Unicode systems store characters in special strings usually defined as BSTRs. These are 16-bit wide multi-byte chars matching a "C" short (VB Integer) data type rather than the usual legacy 8-bit "C" char (VB Byte) data type. The use of 16-bits enables a wider range of characters to be represented including non-English symbols.

Visual BASIC uses Unicode strings internally throughout, although conversion is usually automatic with no intervention being needed. Most DLLs and standard "C" functions use ANSI strings which are roughly equivalent to legacy ASCII strings composed of 8-bit characters.

The BSTR data type can hold both Unicode (multibyte) or ANSI strings. They have data structure with header which records the memory allocation. Standard "C" strings do not; these are simple arrays of type "char" terminated with a single null character. Strings are returned from VBToolbox internally as BSTR data types even when ANSI strings are returned from a function. Visual BASIC is able to perform the necessary conversion automatically during call and return.

VBToolbox mostly works with ANSI/ASCII strings throughout and VB handles these reliably when returned. Where necessary you can also force conversion using the VB **StrConv** function.

Examples:

```
Debug.Print StrConv(GetUserDir(),vbUnicode)
Prints: "C : \ D o c u m e n t s   a n d   S e t t i n g s \ A d m i n"

Debug.Print StrConv(StrToHex("Hello",5),vbUnicode)
Prints: "4 8 6 5 6 C 6 C 6 F "
```

Caution - UPX Compression

The DLL will usually supplied UPX-compressed but **extreme care** should be exercised if re-compressing it.

More importantly, if you intend to use the **LogEvent**() Windows NT/XP+ Event-Logging features you should, **never** repackage the DLL as this will corrupt the event-string resource. Consequently you will not be able to uncompress them properly after compression.

Compressing could cause corruption of your event-log contents when read by the Event-Viewer. If you accidentally compress it then use the command:

UPX -d mslib145.dll

– to fully-decompress it again.

Caution - Thread Safety

This library should **not** be considered thread-safe and no guarantees of thread-safety are offered.

Apparently many aspects of VB5 and 6 and some aspects of SAFEARRAY code are not entirely thread-safe either and the presence of some static variables within "C" functions means the code is not safely re-entrant. Thread-critical code should either call known thread-safe API functions or specific thread-safe code should be written with appropriate locks etc. If in doubt, test thoroughly.

If in doubt, test thoroughly before using with critical code.

Caution - DLLs Are Case Specific!

The links to a DLL interface are case-specific. You **must** use the correct spelling and case when calling DLL functions via Declare statements or the interface will either not work or may work unpredictably. Pay particular attention to the correct function return types.

Caution - This is a 32-bit Library

This is a 32-bit only library. There is no demand for a Visual BASIC 3.0/16-bit version and one will not be produced. Recommended development platforms are VB 4.0, 5.0 and 6.0 but the DLL will work with any language which can reference DLLs such as FreeBASIC or PowerBASIC.

The library should not be assumed to be compatible with Visual BASIC versions prior to version 5.0

The library should be assumed to be incompatible with 64-bit systems.

The library may be compatible with 32-bit Windows emulators such as WINE and possibly ReactOS but it has not yet been tested on those platforms.

Caution - Function Parameters - Use Function Return Values

Avoid using the "Call" syntax on external Functions which return values

Despite the Visual BASIC 5 documentation You should ensure you use the function body return for external code declared using **Declare Function**, rather than relying on a function parameter unless this documentation for that function recommends otherwise. The VB documentation states:

> "If you use either Call syntax to call any intrinsic or user-defined function, the function's return value is discarded."

However, testing has shown there to be a bug in VB5 where failing to use return parameters or using a Declared function as a Sub will cause VB to become unstable and crash on exit or memory regions for variables to be overwritten. This is not a bug in VBToolbox and would apply to any external or API function which returns a value and is called using the "Call" syntax. Debugging such problems would be extremely difficult. The most likely cause of problems is that VB suffers from a buffer-overflow condition. This may not show as a memory-leak but in random instability and unexpected changes in the values of variables.

The following conditions are required for this to happen:

- You are calling a declared external Function rather than a Sub
- You are using Declares rather than a Type Library (TLB)
- You are not retrieving a return parameter OR are not enclosing the function parameter in parentheses OR you are using Call with an external function which returns a value.

- Type library (TLB) versions are not affected as far as can be ascertained
- External routines declared as Sub are not affected, only Functions
- This is not an issue of BSTR v's LPSTR declares (ByVal) declares
- This is not an issue of BSTR* v's LPSTR* (ByRef) declares
- This is not due to problems within the code inside the external DLL - the problem occurs on empty or "(null)" DLL function wrappers

Unless stated in the documentation (e.g. EncryptString), a Function call should be enclosed in parentheses and the return value used, particularly the case of functions handling and returning strings such as StripL(). Functions within this library generally cannot be used as commands (Subs) as with those provided by VB. For example. Wherever practical function parameters have been declared internally within the DLL as "constant" values and are therefore not changed. Usually a copy of the parameter is returned which is allocated by the Windows API which allows VB to destroy and "garbage-collect" the memory.

```
Trim x          ' Permitted by VB with intrinsic VB function but has no effect on variable "x"
StripL x        ' Not permitted by this library -The VB  syntax-checker should block this
Call StripL(x)  ' Not permitted by this library - Function return is not properly used/deallocated

y=Trim(x)       ' Permitted by VB intrinsic function
y=StripL(x)     ' Mandatory for all VBToolbox functions (not Subs)
x=StripL(x)     ' Where you wish to change and then update the value of x
```

Care should be particularly exercised where fixed-length strings are supplied to functions since an entirely new string will be returned by the function body and the original string usually left unchanged.

Caution - Function Parameters - Use Function Return Values (Continued...)

It's worth noting that Visual BASIC doesn't alter the value of parameters within string functions. As an example here is how the Trim() function modifies parameters and returns in VB5.

Example:

```
Dim a As String
Dim b As String

a = "  Hello "
b = Trim(a)
Debug.Print "a=["; a; "]"
Debug.Print "b=["; b; "]"
'    a = [  Hello ] ' Function parameter unchanged
'    b = [Hello]    ' Function return changed

a = "  Hello "
Trim a
Debug.Print "a=["; a; "]"
'   a=[  Hello ]    ' Function parameter unchanged
```

Example Unstable Code

```
Declare Function Foo Lib "libname" (ByVal Value as String) as String

Foo x           ' Crash
Call Foo(x)     ' Crash
y=Foo(x)        ' Safe
```

Example Unstable Code:

```
Dim j As Long
Dim Max As Long
Max = 100000   ' This code will crash due to VB5 bug & not using funct-return
For j = 1 To Max
        If j Mod Max / 10 = 0 Then
                WriteLn j
                Debug.Print PrintR(Join(QSort(StrSplit( _
                        "six, three, one, twelve,6,", ",")))
        Else
                Call QSort(StrSplit("1,two,3,hello,5,yello,zero", ","))
        End If
Next
```

Caution - Visual BASIC and Unsigned Long Values

The Long data-type used in Visual BASIC is a signed type with a range of -2,147,483,648 to 2,147,483,647. Although it offers equivalent long data types "C" also offers an unsigned long type which has a range 0 to 4,294,967,295.

There may frequently be a need to convert hex values outside the signed range into a VB long data type. Hence you may observe large numeric values specified as negative numbers where the leftmost or most-significant-bit (sign bit) is set. Two different hexadecimal values will have also the same signed absolute (abs()) decimal value for a given number. For example:

```
Const HKEY_LOCAL_USER As Long = -2147483647#         ' Hex value is: &H80000001
                                                     ' &h:     8    0 -    0    1
                                                     ' Bit: 8421 8421 - 8421 8421
                                                     ' Val: 1000 0000 - 0000 0001

Debug.Print hex(-2147483647)                         ' Prints 80000001 (signed input)
Debug.Print hextolong("80000001")                    ' Prints 2147483649 (unsigned)
Debug.Print longtohex(-2147483649)                   ' 7FFFFFFF (interpreted as signed)
http://www.google.com/search?q=convert+-2147483649+to+hex   (prints -0x80000001)
http://www.google.com/search?q=convert+0x7FFFFFFF+to+decimal(prints 2 147 483 647)
```

Where there has been a need to accommodate passing unsigned long values into "C" routines a VB Double data type has been used in preference. If there is confusion it may help to deal with hex equivalents for large signed or unsigned numbers wherever possible. As shown above, Google can be used to convert between data types when debugging code.

Caution - VB and "C" (DLL) Integers

If you are developing code where the byte-level of data-encoding is important then you need to bear the following in mind.

The "C" data-type **int** is machine-dependent and it's byte-length is therefore not specified or guaranteed. An **int**. in a Win32 "C" compiler such as MSVC is 8 bytes long (the same as a **long**) but the VB **Integer** data-type is only 4 bytes long. If you are developing "C" or "C++" code you must ensure you use either a **short** or **unsigned short** data type which is *guaranteed* to be 4 bytes long.

VB was designed to handle signed values for each given integer type, functions which require the handling of unsigned values, such as hex conversion routines, are declared to pass the next-highest data type in order to ensure flagging the leftmost "sign" bit does not corrupt values.

Remember: Visual BASIC **Integer** = "C"/C++ **short**

Caution - String Parameters - Use ByRef in Declares Only Where Specified

You must usually call DLLs using strings using **ByVal** and ***not*** **ByRef** unless the DLL has specially been written to accept **ByRef** calls. The correct declarations are given in mslib145.bas. VB supplies a "pointer to string" when you pass **ByVal**. If you pass **ByRef** then you will be passing a pointer to a pointer (**char****). Which is usually NOT what you want.

Strings which return "full binary" characters (0..255) outside the normal printable-range may, in some cases need, to be trimmed using the supplied Visual BASIC **VBStr()** routine. Check each function definition for more information.

A brief summary of ByRef v's ByVal as translated to the "C" interface by VB is as follows:

VB Type	**ByVal**	**ByRef**
Byte	*char*	*char**
Integer	*short*	*short**
Long	*long*	*long**
Double	*double*	*double**
String	*char**	*char***

A call ByRef is used in a few cases with VBToolbox where a value other than String is changed and returned. e.g. StringToBMP, BMPInfo, BGRSplit etc.

Example VB/C-DLL functions compared:

You should see what happens in any called "C" module when you confuse ByVal and ByRef. Get it wrong and indirection will fail. Your app will get a pointer to a value when it expects a value or vice-versa. Any subsequent dereferencing of a pointer or addressing a value may cause a GPF.

VB	Public Declare Sub FooConstInt Lib "some.dll" (ByVal i as Integer)
"C"	void _stdcall FooConstInt(short i); // L may not be changed (copy)
VB	Public Declare Sub FooChangeInt Lib "some.dll" (ByRef i as Integer)
"C"	void _stdcall FooChangeInt(short* i); // L may be changed (pointer)
VB	Public Declare Sub FooByVal Lib "some.dll" (ByVal s as String)
"C"	void _stdcall FooByVal(char* s); // Pointer to C string
VB	Public Declare Sub FooByRef Lib "some.dll" (ByRef s as String)
"C"	void _stdcall FooByRef(char** s); // Pointer to an array of C string pointers

Notice also that the VB Integer data type is a "short int" in "C" and not "integer".

DLL Functions Which Return String Values

Although NULL-terminated strings are no longer returned by VBToolbox this section is retained for reference.

"C" does not have a native "string" data type as such. Strings in "C" are simply arrays of characters. No record is ever kept of the length of a string by the "C" compiler. Instead the convention is that all such "strings" are terminated with a NULL (0, or '\0') character, as in …

```
"The cat sat on the mat0"
```

The absence of this terminator is a frequent reason why many C and C++ programs crash catastrophically after losing track of strings and accessing areas of memory which the program does not "own".

VB handles strings in a different way, keeping an exact record of how long the strings are and, thus, doesn't need the terminating NULL character. Consequently if C/C++ strings are returned from a DLL to VB and exact string lengths aren't calculated then the null character needs to be stripped off to avoid a longer string than expected being processed by VB. A functions is provided for this conversion process where necessary in Visual BASIC - VBStr() and it's alias StripTerminator().

These two functions are defined in the mslib145.bas module which also includes the required DLL interface definitions and must therefore included with all of your VB projects.

> *Most functions from V1.11 onward will no longer require the use of StripNulls() or VBStr() etc. to trim terminating NULL characters. Most functions have been rewritten for exact-memory-allocation where Visual BASIC is allowed to take care of string-deletion and "garbage-collection"*

See the section on Visual BASIC Wrapper Functions and the notes given for each function for more information.

Possibly Unnecessary Function Exports

Some may consider many of the Windows API functions unnecessary to include as they may be just as easily be utilised via a direct declare to the relevant Windows DLL.

The reason such functions are included is so they may be exported via the Type Library (TLB) file without the necessity of declare statements at all. They are made available via declares merely for the sake of completeness.

Additionally where DLL wrappers are places around external API calls which could otherwise be called directly without the VBToolbox DLL they may include additional error or parameter checking which can make your code more stable (checking for null pointers, invalid or empty strings etc.).

Duplicate Name Conflicts When Calling DLLs

If the given names conflict with those of another function or DLL an “Alias” can be declared.
You can rename a DLL and call it within VB using any name you like.

For example, if you want to use a function exported from the VBToolbox DLL called "EncryptFile64" using the function name “Encrypt64” then simply change or add an "Alias" declaration as follows (fictitious example):

```
Private Declare Function Encrypt64 Lib "mslib145.dll" Alias
"EncryptFile64" (ByVal FileName As String, ByVal Buffer As String) As
String
```

Correct DLL Function Calling Conventions and Visual BASIC IDE Issues

It is vital that you adhere to proper calling conventions when calling external DLLs. During testing it was found to be possible to create unstable local variables in one location within VB if an incorrect function call was made earlier on in program code.

For example, the following illegal function call (command or VB "Sub"-style) method of calling an external function will not be properly checked. was made during testing ...

```
StripL Temp, " h"
```

This resulted in no problems and should have been syntax trapped by the IDE. However, later on during testing an integer was declared and it was found that the value of this integer was changing randomly within the program code and this was also traced later to be happening with each and every call to **Debug.Print**. It was clear from this that the IDE and the VB variable allocation table had been corrupted, possibly by the string return from **StripL**() not being handled or released correctly.

The correct syntax to use when calling external functions should be to either use "Call" or make a proper function call which assigns the return value to a variable ...

```
Call StripL(Temp," h")
stringvar=StripL(Temp," h")
```

If this form is used the IDE will properly check for syntax errors.
An example of this bug is shown below...

Code:

```
StripL Temp, " h"
Dim X as Integer
X=10
Debug.Print X
Debug.Print X
Debug.Print X
Debug.Print X
```

VB Immediate Pane Output Result:

Notice that X varies randomly between each call, almost certainly due to memory-corruption ...

```
33
0
-1
-1
-1
```

See cautions relating to the use of function returns

Visual BASIC, Windows XP and Data Execution Prevention (DEP) Issues

Being released prior to Windows XP, the Visual BASIC 5 IDE appears to require a DEP exception to be configured. DEP (Data Execution Prevention) was a new feature provided with XP Service Pack 2 which is designed to prevent code being executed from areas of memory flagged as "NX" (no execute). By default DEP is enabled only for essential Windows applications and services.

If you enable DEP for all programs then you will need to include Visual BASIC 5 in the exceptions list or you may experience the following error. Note that this error is due to the VB IDE and NOT due to VBToolbox.

You can replicate the problem in XP as follows.

1) Enable DEP for all programs by going to Start-> Settings-> Control Panel-> System-> Advanced Tab-> Performance (Settings)-> Data Execution Prevention Tab.
2) Then select "Turn on DEP f or all programs and services except those I select". (Ensure Visual BASIC is not in the exceptions list to test this problem).
3) Launch the Visual BASIC IDE and crate a new, EXE project. It should open with one form (Form1)
4) Enter or edit the Form1 code as follows...

```
Private Sub Form_Load()
    Debug.Print "Running..."  'Breakpoint set here if rqd.
    Call x
End Sub

Private Sub x()
    'Does nothing
End Sub
```

5) If you wish, you can set a breakpoint at the Debug.Print "Running..." line (it will never be executed)
6) Save the project in a new folder, called, say, "empty-project"
7) Click run. You should get a DEP crash.

- You can also replicate the problem by entering only the name of a non-existent sub or function (which should be trapped by the IDE when run). This also causes a DEP crash.
- These are all VB/XPSP2 interaction issues and not related to VBToolbox.
- Compiled EXE programs still work fine, including those created with VBToolbox

DEP Problem Workaround

Either re-enabled the default XP DEP setting (only essential Windows programs and services) or configure Visual BASIC as an exception to DEP. You can use the prompt offered by the crash-handler. Note that after changing DEP settings you must exit and re-load any open VB5 IDE projects.

Console Functionality

Unlike VB6, VB5 doesn't natively produce console applications which can attach to and write to any currently open console (CMD) window. This functionality is provided here by allowing your VB executable process to open it's own console window.

Your program will not write to any currently open console window when first compiled. This is not a bug it is by design due to the limitations of the VB5/6 compiler. The EXE may be re-linked with LINK.EXE as a full-console binary (see the Console section of this manual). Bear in mind it's the EXE which needs to be re-linked despite functionality being provided by the VBToolbox DLL.

Server CGI Applications

Despite not writing to any currently open CMD.EXE console window the console functionality works absolutely fine with Server CGI scripting allowing you to develop back-end CGI applications using Visual BASIC 5, 6 or any similar language. Console processes are spawned at the server end to provide input and output to stdout and stdin and destroyed when the program terminates. You should, however always use CloseConsole to forcibly close any console window you have opened. Functionality has been tested using Apache/Win32.

A small Win32 CGI test program and VB project/source-code are available on the VBToolbox site.

Specific HTTP and CGI functionality is planned for implementation at a later date.

Intended Language and O/S Platforms

This project was intended for use with legacy Visual BASIC 4.0, 5.0 and 6.0. You should also be able to use the libraries successfully from MSOffice, VBA, Visual FoxPro, FreeBASIC etc. or any other language which accepts declares and interfaces for 32-bit Windows DLLs. Generally it should be assumed that this library will work satisfactorily with Windows XP or Vista only. Feedback from testing on other O/S versions including WINE is always appreciated.

This version will have been tested and should work satisfactorily with the following versions of Windows:

- Windows NT Workstation 4.0 (SP6a)
- Windows 2000 (SP3+)
- Windows XP (SP2+)
- Windows 7

Other DLL Libraries You Can Use

You can use any installed DLL if you know the correct interface method and are able to build up "legal" declares for it. You can use the entire "C" function library from MSVCRT*.DLL (if installed) and a list of exported functions for this library is included at the end of this manual

Why do you call it "BASIC" instead of "Basic"?

BASIC is an *acronym*, or letter-abbreviation for Beginner's All-Purpose Symbolic Instruction Code, whereas Pascal, Fortran, C++, Pilot, Java etc. are not.

Bug Reporting

Please report any bugs via the website giving full details of the problems and, if possible a sample of the code involved. As this is a *freeware* product there is NO WARRANTY and I can't guarantee I'll be able to reply directly. Well-documented bug-reports are taken seriously and I will endeavour to rectify any "bugs" or address any requests for new features in the next release.

Website: http://vbtoolbox.kerys.co.uk

VBTooolbox Installation

Installation Procedure

- Copy the supplied DLL to your Windows system directory. This will be something like …

```
C:\WINNT\SYSTEM32
C:\WINDOWS\SYSTEM32
```

For NT, Windows 2000 or XP

or for legacy Windows 9x systems …

```
C:\WINDOWS\SYSTEM
C:\WIN95\SYSTEM
C:\WIN98\SYSTEM
```

Alternatively you can leave the DLL in the application directory if you only want to do testing or place it in any folder which is in your current PATH variable.

- You may rename the DLL if you wish. Avoid multiple copies of the DLL on your system
- For the ANSI version, copy the MSLIB145.BAS module to convenient a folder such as \VB\INCLUDE - Drag and drop the file onto your project list for the project you want to include the library for
- For the Unicode/TLB version, ensure you browse to the location of MSLIB146.TLB and include it as a project reference
- That's it!. The functions are now available.

Installation Troubleshooting

- Note that console I/O may only work in properly in Windows NT, XP and W7 - it is untested in Win9x which is now obsolete and has a declining user-base. Windows 3.x is not supported.

- If you have problems then check you have downloaded the latest copies of both the BAS or TLB module and the matching DLL.

- Check that you have them *both* TLB/BAS module and DLL module installed as a matched pair as future updates may change the interface specifications.

- If you can't rename or delete the DLL to install a new version, exit VB or any application which uses the DLL as these will "lock" the DLL. If you still can't delete or rename then reboot the PC and ensure that no application which uses the DLL is loaded at startup.

- Ensure that you only have *one* copy of the DLL in the currently pathed directories, particularly if you have different versions of the DLL installed.

Visual BASIC - Application Setup Wizard - Install/Setup - Troubleshooting

There is a bug in the Visual BASIC 5 setup program which can cause setup to fail under certain circumstances. If you run SETUP.EXE from a location where the path contains non-alphanumeric characters then the legacy Windows 3.x code which appears to form the basis of the setup program gets confused and will refuse to install your completed project. You will receive the error:

"Invalid command-line parameters. Unable to continue"

and setup will exit. If this occurs check that it has not been launched from a location which contains underscore (_) or hyphen (-) characters in any part of the directory/folder/path name. If you still have

problems copy the setup disk set image (Disk1, Disk2 etc...) to a root folder such as c:\setup and try again from there. You may find it will work perfectly.

Function Interface List

The functions are grouped together in the manual according to several categories. This taxonomy is by no means definitive or concrete. Some data-conversion functions might be viewed as maths functions for example. If in doubt, refer to the index at the rear of this manual.

DLL Management Functions

Function - LibDate

Declare: Private Declare Function **LibDate** Lib "mslib145.dll" () As String

Purpose: Used to check if the correct release-date version of the DLL is loaded or installed.

Returns: Returns a string in ISO date format as "YYYYMMDD"

Function - LibName

Declare: Private Declare Function **LibName** Lib "mslib145.dll" () As String

Purpose: Used to check the compiled filename of the DLL which has been loaded

Returns: Returns a string matching a filename: e.g. "MSLIB145.DLL". No path value will be returned and this is the compiled filename which could, potentially be renamed. Use GetDLLFileName to retrieve the actual loaded filename and path location.

See also: **GetDLLFileName**

Function - LibTime

Declare: Private Declare Function **LibTime** Lib "mslib145.dll" () As String

Purpose: Used to check if the correct release-time version of the DLL is loaded or installed.

Returns: Returns a string in 24-hour time format as "HH:MM:SS" indicating the time the DLL was compiled.

Function - LibVersion

Declare: Private Declare Function **LibVersion** Lib "mslib145.dll" () As Long

Purpose: Used to check if the DLL is loaded or installed. Can be called safely with the following routine which is included in the MSLIB145.BAS file.

Returns: Returns an integer with the version multiplied x 100 (e.g. v1.03 = 103)

Notes: This function was originally exported as "_libversion" (case specific)
Divide the return by 100 to get the version number

Example: Checks if the DLL functions are installed and available so you can decide whether or not to use them in your program. If the DLL is not installed then IsDLLInstalled records the error event as a "False" value.

```
Private Function IsDLLInstalled() As Boolean
    Dim r As Long
    IsDLLInstalled = True

    On Error GoTo NoDll
    r = LibVersion()
    On Error GoTo 0
    Exit Function

NoDll:
    'This is triggered if the DLL is not available or the function
    'has not been exported (i.e. Wrong DLL version)
    IsDLLInstalled = False
    Resume Next
End Function
```

Suggested use:

```
Public DLLInstalled as Boolean
…
Sub Main()
        DLLInstalled=IsDLLInstalled()
End Sub
```

You can also use the global version string within your program to check for compatible versions of this DLL using DLLVersion(). This returns a string in the format "xx.x" e.g. "1.02"

```
Public Function DLLVersion() As String
    Dim VerNum As Integer
    Dim Temp As String
    On Local Error Resume Next
    Temp = "0.00"
    VerNum = LibVersion()
    Temp = Format(VerNum)
    Temp = Left$(Temp, 1) & "." & Right$(Temp, 2)
    DLLVersion = Temp
End Function
```

Function - LibUnicode

Declare: Private Declare Function **LibUnicode** Lib "mslib145.dll" () As Boolean

Purpose: Used to check if the loaded DLL is the ANSI or Unicode version. This does not indicate full Unicode support is available in any given version. The ANSI version does not support a Type Library (TLB) and requires Declare statements. The Unicode version is supported with a TLB file but may also use Declare statements.

Returns: Returns a Boolean True or False value

String Handling Functions

The string-handling collection includes a number of routines which can simplify the amount of external calculations and pointer mathematics involved. These involve simple ways to split and join strings. For example, splitting functions include CSVSplit, StrSplit, Tokenise, BracketStr MidCharStr, MidStrStr and joining functions include Join (as per VB6), InsertString

When handling returns from String() array or Variant() String array functions it will often be desirable to check whether the String or Variant array is empty before calling other code such as UBound or LBound etc. In such cases the returned value should be tested using the VB **IsArray**() function rather than **IsEmpty**() or **IsSet**(). This is because a valid variant is always returned to enable function returns to be chained.

Function - AddString

Declare: Public Declare Function **AddString** Lib "mslib145.dll" (ByRef a As String, _
ByVal b As String) As String

Purpose: Functional replacement for the concatenation operator "&" which uses the Win32 System API to allocate memory rather than restricted VB string space.

This function is only suitable for non-binary ASCII (ANSI) "text" strings which have no embedded NULL characters (0x00). Use AddBinaryString for strings which may contain embedded NULL characters.

Notes: Strings which are returned are "garbage collected" by VB
Parameter "a" is freed on successful concatenation and set to NULL
Maximum transient memory allocation is (2 x a) + b
Currently no error return indicates an out-of-memory condition. You can use GetError() to check this

Notes: If the 2nd parameter is NULL a new string containing the 1st parameter is returned

See Also: **AddBinaryString, AddHugeBinaryString, AllocString, GetError**

Function - AddBinaryString

Declare: Public Declare Function **AddBinaryString** Lib "mslib145.dll" (ByRef a As String, _
ByVal b As String, Optional ByVal Length As Long = 0) As String

Purpose: Functional replacement for the concatenation operator "&" which uses the Win32 System API to allocate memory rather than restricted VB string space.

This function is suitable for binary or ANSI strings which may have embedded NULL characters (0x00). Consequently, for binary strings, the length of the string to append must be specified using the "Length" parameter.

Notes: Unless the 2nd parameter has been allocated by API memory allocation AddBinaryString requires that the precise length of the 2nd parameter be given. This is particularly the case where string literals are concerned. The caller is responsible for the accuracy of the length value. No checks are made.

If parameter "length" is zero then the entire 2nd parameter string will be copied
Strings which are returned are "garbage collected" by VB
Parameter "a" is freed on successful concatenation and set to NULL
Maximum transient memory allocation is (2 x a) + b
Currently no error return indicates an out-of-memory condition. You can use GetError() to check this
There is a practical memory limit of about 200 mb on the total size of input strings dependent on available system memory (See AddHugeBinaryString)

If the 2nd parameter is NULL a new string containing the 1st parameter is returned

See Also: **AddString, AddHugeBinaryString, AllocString, GetError**

Function - AddHugeBinaryString

Declare: Public Declare Function **AddHugeBinaryString** Lib "mslib145.dll" (_
ByRef a As String, ByVal b As String, Optional ByVal Length As Long = 0, _
Optional ErrCode As Integer = 0) As String

Purpose: Functional replacement for the concatenation operator "&" which uses the Win32 System API to allocate memory rather than restricted VB string space.

This function is suitable for very large binary or ANSI strings of up to say 300 megabytes or more which may have embedded NULL characters (0x00). Consequently the length of the string to append must be specified using the "Length" parameter.

Notes: IMPORTANT – As this function attempts to concatenate both input strings in-memory then both input strings are released (deleted) from memory and the allocation pointers set to NULL. Any resulting string which has been successfully concatenated is returned by the function return. In order to achieve this both input strings are passed *by reference* and not *by value*.

Return memory is allocated using the SysAllocString() API and will be managed by VB. SysFreeString() should not be called with a null or invalid pointer.

If the strings are over 100 megabytes in size then the function will try to use an intermediate temporary file on the hard drive to concatenate the strings. If the disk is full this function will fail and return NULL for all string inputs and returns.

If parameter "length" is zero then the entire 2nd parameter string will be copied

This function is NOT exported via the TLB Unicode interface

Since there are many potentials sources of failure when concatenating even a single character to a huge string an optional error return parameter is provided which will return the following error codes -

0	No error
2	File not found
3	Can't open file
4	Can't allocate memory
5	Insufficient disk space
6	Failed to create a free temporary filename

Codes may change in future versions

See Also: **AddString, AddBinaryString, AllocString**

Function - AllocString

Declare: Public Declare Function **AllocString** Lib "mslib145.dll" (ByVal l As Long, _
Optional ByVal BlankChar As Byte = 0) As String

Purpose: Allows extremely rapid allocation of strings using direct Windows API calls. Has the same functionality as String$ but under some circumstances this may be faster.

See also: **FillString, AddString, AddBinaryString, AddHugeBinaryString**

Function - ArgFound

Declare: Public Declare Function **ArgFound** Lib "mslib145.dll" (ByRef v As Variant,
ByVal s As String, Optional ByVal IgnoreCase As Boolean = True)
As Boolean

Purpose: Tests a string array stored as a Variant to see if an argument LValue is present for a pair of values. Each of the values in the array will be in the form "x=y" pair. e.g. "Value=ten".

The function does not test to see if an RValue exists, just for the presence of the Lvalue. For the pair "Value=ten" the presence of the string "Value=" is checked.

The search is case-significant by default but can be ignored by setting IgnoreCase

Example: For the 3-item array below stored in Variant V from using Tokenise() or GetArgs()

```
"A=1"
"B=2"
"C=3"

Debug.Print "Argument B Found?="; ArgFound(V,"B")
```

Result: `"Argument B Found?=True"`

See also: **GetArgs, GetCGIArgs, ArgVal, Tokenise**

Function - BracketStr

Declare: Public Declare Function **BracketStr** Lib "mslib145.dll" (ByVal s As String, _
ByVal p As Long, _
Optional ByVal StripBrackets As Boolean = False) As String

Purpose: Tests for a matched bracket pair within a given string or string expression and returns the bracketed substring if and only if the bracket pairs match.

Notes: The bracket-type is automatically detected and matched.
The following bracket-types are automatically-recognised: () [] {} <>
The input character position location is a base 1 value
On error or bracket not matched NULL is returned otherwise the bracketed substring is returned. The bracketed expression is returned complete with enclosing brackets unless the optional StripBrackets flag is set to True
Use FindClosingBracket to retrieve only the location of the closing bracket

See also: **FindClosingBracket**

Function - ArgVal

Declare: Public Declare Function **ArgVal** Lib "mslib145.dll" (ByRef v As Variant, ByVal s As String,Optional ByVal IgnoreCase As Boolean = True) As String

Purpose: Tests an array to see if an argument LValue is present for a pair of values.

Notes: If the LValue is found then the RValue (if any) is returned. The function may return NULL ("") if the RValue is absent. Each of the values in the array will be in the form "x=y". e.g. "Value=ten".

The search is case-significant by default but can be ignored by setting IgnoreCase

Important: You should call **ArgVal** before calling **URLDecode** on any CGI query string. The reason for this is that the ampersand character (&) is used to delimit the query string and is therefore used to "tokenise" or split the string into an array. When the ampersand is used in text the character is encoded as %26 by the browser or web server. You should call URLDecode only after the array has been split otherwise the character will be wrongly interpreted as a delimiter character.

Example: For the 3-item array below stored in Variant V using Tokenise() or GetArgs()

```
"A=1"
"B=2"
"C=3"
```

Debug.Print "Argument Value for B ="; ArgVal(V,"B")

Prints out: "Argument Value for B =2"

See also: **GetArgs, GetCGIArgs, ArgFound, StrSplit, Tokenise**

Function - CommaStr

Declare: Public Declare Function **CommaStr** "mslib145.dll" (ByVal S As String, ByVal DecimalPrecision as Integer) As String

Purpose: Takes a string and formats it with commas at every three digits. Trailing values after the decimal-point are simply truncated at whatever value is specified for *DecimalPrecision*. No rounding is performed. *No checks are made to see if the string contains valid numeric values.* Use Comma() to safely format numeric values.

Example: `CommaStr("12345678.901234",2)  returns "12,345,678.90"`

See also: **Comma**

Function - Comma

Declare: Public Declare Function **Comma** "mslib145.dll" (ByVal Value as Double, ByVal Optional Decimals as Integer=2) As String

Purpose: Takes a number and returns a formatted string separated by commas. Trailing values after the decimal-point are simply truncated at whatever value is specified No rounding is performed *and no checks are made to see if the string contains valid numeric values.* Use CommaStr() to format string values. However, under tests it seems that VB (5 at least) automatically converts numeric values to strings.

Example:
```
Comma(12345678.901234,2)      returns "12,345,678.90"
Comma("12345678.901234",2)    also returns "12,345,678.90"
```

See also: **CommaStr**

Function - CSVSplit

Declare: Public Declare Function **CSVSplit** Lib "mslib145.dll" (ByVal s As String, _
Optional ByVal ArrayBase As Long = 0, _
Optional ByVal SeparatorCharVal As Byte = 44) As Variant

Purpose: Parses and splits a line of industry-standard Comma Separated Value (CSV) items These are items delimited by comma characters (Chr$(44)) and optionally with each field delimited by double quote characters (Chr$(34)).

Notes: A zero-based array is returned by default. You may change this with the ArrayBase parameter. You may change the delimiter character by supplying the ASCII number of a replacement.

Whitespace outside quoted fields is stripped but not within data-fields

Double-quote characters are stripped-off fields. Where excess double-quote characters exist within a field only the outermost pair are stripped. Thus, quotes which form part of a data field are left intact. e.g.

""This string contains quotes"","This doesn't" will transform to the array:

[0] => "This string contains quotes"
[1] => This doesn't

The function will do it's best to handle misformed fields such as ...,"a"b,... and mismatched double-quotes.

The function times at about 8.3 microseconds per call for a 10 element string array but timings depend how much work has to be done on complex formatting.

Example:

```
' Input CSV String "s" is: "one,2,"Three","four"a,b,c"
' Note that PrintR encloses displayed strings in quotes
PrintR(CSVSplit(s,Asc(","))
```

Result:

```
ByVal: VT_0x2008: Array of Variant->String(6)
(
        [0] => "one"
        [1] => "2"
        [2] => "Three"
        [3] => "four"a"
        [4] => "b"
        [5] => "c"
)
```

See also: **PrintR, StrSplit, StrToken**

Function - ElementCount

Declare: Public Declare Function **ElementCount** Lib "mslib145.dll" (
ByRef v As Variant) As Long

Purpose: Returns a count of the number of elements (if any) in a Variant array

Notes: UBound will fail with an error on on empty Variants or empty Variant arrays – which isn't particularly useful behaviour if you want to use UBound intelligently. ElementCount safely returns 0 for all types of empty variant and the count of elements for non-empty Variant arrays.

Example:

```
Dim V as Variant
For i=1 To 10          ' Dynamically-redimension and add items
        ReDim V(ElementCount(V)+1)
        V(i)=i
        Debug.Print "v("; i; ")="; v(i)
Next
```

See also: **GetArrayDimensions, IsArray (VB)**

Function - Expression

Declare: Public Declare Function **Expression** Lib "mslib145.dll" (ByVal s As String, _
Optional ByRef ErrorCode As Integer = 0) As Double

Purpose: Evaluates a complex mathematical expression held in a String, returning the result as a Double. Expression is expected to be a useful debugging tool or for use as a handy calculator function.

Notes: An error return value is provided, this should be used to detect problems with the expression and safely handle any required actions.

There is no operator precedence (BEDMAS or BOMDAS). The expression is evaluated from left to right. Precedence must be enforced using standard curved brackets "()"

All values are computed internally as Doubles wherever possible. Where integer values must be used such as with bitwise operators numbers are first coerced to signed 32-bit Long data types. Be aware that this may cause truncation and loss of precision. Hex values are always evaluated initially as unsigned.

Due to the function being based on "C" libraries there may be a slight difference in precision between Expression and the VB Immediate pane evaluation of the same expression, particularly with very large numbers using "scientific precision".

Boolean values return 1 or 0 internally. An internal Boolean return may be interpreted by Visual BASIC as either 0 (False) or -1 (True)

Comparison operators such as ">" or >=" return a Boolean 0 or 1 internally

Hexadecimal values are permitted using either the VB "&h" prefix or "C"-style "0x" prefix – e.g. &h100 or 0x100. Be aware that since hex values are intended to be mainly used for masking and bit-shifting rather than arithmetic they are held as unsigned values.

Binary values are specified using an &b prefix, e.g. &b111 (decimal 7)

Roman numeral values are specified using an &r prefix, e.g. &rVII (decimal 7)

All non-decimal formats are converted to decimal format before evaluation

The "pow()" (power) operator is an **infix** operator. i.e. it applies it's l-value to the current accumulator and it's r-value to the next value to be met. It cannot be used in prefix/function style – e.g. pow(x,y). It must be used as x pow y There is also a limit of 264 to the exponent part of the power operator.

Operators: The following operators and keywords are understood by Expression

Symbol	Keyword	Description
+		Addition (plus)
-		Subtraction (minus)
-		Negation
*		Multiplication
/		Division
%	**mod**	Modulus – (Remainder after integer division – x mod y where y>0)
^	**xor**	Bitwise xor
~	**not**	Bitwise not
&	**and**	Bitwise and
|	**or**	Bitwise or
<<	**shl**	Bitwise shift left
>>	**shr**	Bitwise shift right
>	**gt**	Boolean greater-than
<	**lt**	Boolean less-than
<=	**le**	Boolean less-than or equal-to
>=	**ge**	Boolean greater-than or equal-to
==	**eq**	Boolean equivalence
!=	**ne**	!= Boolean not equal to
	pow	x pow y – Infix operator – raise x to the power of y (max y value is 246)

Errors: Expression returns error codes as follows to indicate problems during evaluation One one of the following cases an error return is set and the 0.0 is returned via the main function body.

1	Syntax Error (The expression was badly-formed)
2	Division by Zero
3	Multiple Operators Met
4	Unexpected Bracket (Brackets did not match properly)
5	Number is Too Large to Evaluate
6	Invalid Expression (The expression was not valid)
7	Out of Heap Memory
8	Overflow (Resulting number was too large to hold)
9	Hex Digit Overflow
10	power (pow) Range Error
11	Roman number too large
12	Binary number too large
11	Unknown Error

Example:

```
Debug.Print Expression("(((((14310+564)/6846)^2042)+5366)/7944)+9522
Debug.Print (((((14310+564)/6846) xor 2042)+5366)/7944)+9522
```

Result:

```
9522.9322759315        ' VBToolbox Expression() function
9522.9322759315        ' VB Immediate pane
```

See also: **BracketStr, FindClosingBracket, MatchBrackets**

Function - FillString

Declare: Public Declare Function **FillString** Lib "mslib145.dll" (ByVal s As String, ByVal CharToUse As String) As String

Purpose: Fills a string with a given character very rapidly. Useful on extremely large strings. A quick way of erasing of blanking a string.

Example:
```
FillString("1234","Hello!")  returns the string  "HHHH"
FillString("The quick brown fox","*") returns "******************"
```

Notes: To simplify the interface a string is accepted for the character to use but only the first character is used, the remainder (if any) are ignored.

See also: **AllocString**

Function - FindClosingBracket

Declare: Public Declare Function **FindClosingBracket** Lib "mslib145.dll" (_
ByVal s As String, ByVal p As Long) As Long

Purpose: Matches a bracket pair, including nested brackets and return the character-position of the closing bracket.

Notes: The bracket-type is automatically detected and matched.
The following bracket-types are automatically-recognised: () [] {} <>
The input character position location is an array base 1 value
On error or bracket not matched zero is returned. Otherwise a base 1 value indicating the location of the closing bracket within the entire string is returned.
Use BracketStr to retrieve the bracketed expression or substring

See also: **BracketStr**

Function Filter

Declare: Public Declare Function **Filter** Lib "mslib145.dll" (ByRef V As Variant, _
ByVal FilterString As String, _
Optional ByVal Include As Boolean = True, _
Optional ByVal CompareMethod As Integer = 0) As Variant

Purpose: An emulation of the Visual BASIC 6 Filter() function

Filter takes a String array or a Variant array of Strings and returns a new array based on the criteria specified. Using the Include parameter you may choose to either include or exclude elements based on the FilterString parameter. You may also perform either a case-sensitive or case-insensitive search.

Filter takes a Variant argument, String() arrays will be cast by VB as Variants. This enables input to be chained from other VBToolbox functions. A Variant String array is returned which may be chained into other VBToolbox functions such as PrintR

Parameter CompareMethod is either vbCompareBinary or vbCompareText

Example:

```
' Exclude the word "the" from the results
Debug.Print PrintR(Filter(Tokenise("the quick brown fox jumps over _
the lazy dog"," "),"the",false,vbTextCompare ))
```

Result:

```
ByVal:Array     Variant->String(7)
(
        [0] => "quick"
        [1] => "brown"
        [2] => "fox"
        [3] => "jumps"
        [4] => "over"
        [5] => "lazy"
        [6] => "dog"
)
```

See also: **PrintR, StrSplit, Tokenise**

Sub - GetArgs

Declare: Public Declare Function **GetArgs** Lib "mslib145.dll" (ByVal s As String,
v As Variant) As Integer

Purpose: Converts Command$ (when used as the parameter) or any other string into an array of individual arguments akin to the argv[] values present in "C". The function Tokenise() is used to split the string into an array.

Returns the number of elements in a string-array which is returned by means of a Variant (v) in the function body. Note that the array will always be "base 1", that is, the first element will be at

Example:
```
Command$="one two three"
Dim Argc as Integer
Dim i as Integer
Dim Argv as variant
Argc=GetArgs(Command$, Argv)
For i=1 to Argc
       Debug.Print "Argv["; i ; "] is "; Argv(i)
Next
```

Result:
```
one
two
three
```

See also: **Tokenise, ArgFound, ArgVal**

Function - GetArrayCount

Declare: Public Declare Function **GetArrayCount** Lib "mslib145.dll" (_
ByRef V As Variant) As Long

Purpose: Retrieves the number of elements in a String() array or Variant() String array, held either ByRef or ByVal. Note that this may include NULL, Empty or "non-allocated" elements.

Notes: You may use the PrintR() function to manually-inspect the contents of an array and determine whether elements are empty or not

GetArrayCount accepts a Variant parameter and may be chained to the result of other String array functions.

See also: **GetArrayDimensions, ElementCount**

Function - GetArrayDimensions

Declare: Public Declare Function **GetArrayDimensions** Lib "mslib145.dll" (ByRef V As Variant) As Long

Purpose: Retrieves the dimensions of a Variant array, held either ByRef or ByVal

Example:
```
Dim U as Variant
Dim V(10,2) as Variant
Dim W(4) as Variant

Debug.Print "The dimensions of U are "; GetArrayDimensions(U)
Debug.Print "The dimensions of V are "; GetArrayDimensions(V)
Debug.Print "The dimensions of W are "; GetArrayDimensions(W)
```

Result:
```
The dimensions of U are 0
The dimensions of V are 2
The dimensions of W are 1
```

Notes: A string-array has a dimension of 1, an uninitialised Variant or one where no array-dimensions have been specified has a dimension of 0

See also: **GetArrayCount**

Function - GetFileExt

Declare: Public Declare Function **GetFileExt** Lib "mslib145.dll" (ByVal s As String) As String

Purpose: Returns the filename-extension (filetype) part of a path string as "name[.typ]"
The dot is included in the returned string
If a path backslash character appears after the filetype then an empty string is returned.

Example:
```
Debug.Print GetFileExt("c:\windows\system32\calc.exe")
```

Result:
```
".exe"
```

See also: **GetFileName, GetNormalisedPath**

Function - GetFileName

Declare: Public Declare Function **GetFileName** Lib "mslib145" (ByVal s As String) As String

Purpose: Returns the filename part of a path string as "name[.typ]"
The filename need not have a filetype. If no filetype is found then the last path segment will be assumed to be the filename.

Example:
```
Debug.Print GetFileName("c:\windows\system32\calc.exe")
```

Result:
```
"calc.exe"
```

See also: **GetFileExt, GetNormalisedPath**

Function - InChrRev

Declare: Public Declare Function **InChrRev** Lib "mslib145.dll" (ByVal sMyString As String, ByVal iChar As Integer) As Long

Purpose: Finds the position of a character (passed as an integer) in a string searching from the end of the string in reverse direction.

Function - InChr

Declare: Public Declare Function **InChr** Lib "mslib145.dll" (ByVal sMyString As String, ByVal iChar As Integer) As Long

Purpose: Finds the first instance of a character (passed as an integer) in a string searching forwards as with InStr()

Function - InsertString

Declare: Public Declare Function **InsertString** Lib "mslib145.dll" (ByVal s As String, _
ByVal NewString As String, _
Optional ByVal InsertPos As Long = 1) As String

Purpose: Inserts one string into another at any position up to the end of the first string

Notes: The insertion-point is specified as a "base 1" value, i.e. the first character is 1, the 2nd is 2 etc. It is not permitted to insert at a position any higher than 1 character after the end of the string (the string length). Inserting after the end of the string effectively concatenates the two strings together.

Original string and string-to-insert are not changed by this process. The changed string is returned via the function-body.

Function - InStrI

Declare: Public Declare Function **InStrI** Lib "mslib145.dll" (ByVal s As String, ByVal search As String) As Long

Purpose: Equivalent to the built-in VB function "Instr" except that a starting position may not be specified and it is case-insignificant.

Function - InStrRev

Declare: Public Declare Function **InStrRev** Lib "mslib145.dll" (ByVal sMyString As String, ByVal Search As String) As Long

Purpose: Finds the LAST instance of a string within another. Searches from the end of the string backwards to the start of the string.

Function - IsAllChar

Declare: Public Declare Function **IsAllChar** Lib "mslib145.dll" (ByVal s As String, ByVal TheChar As String) As Boolean

Purpose: Rapidly find if a string comprises of a single character. A quick way of finding out if a string is all blank (spaces) etc.

Examples:
```
IsAllChar("","") returns  True
       (the given string is all "" - same as IsNull)
IsAllChar("","x")  returns False
IsAllChar("x","")  returns False
IsAllChar("Hello","H")  returns False
IsAllChar("HHHH",H") returns True
IsAllChar("HHHH",Hello")  returns True
```

Notes: In order to simplify calling the function, a string is accepted as the 2nd parameter rather than a char (Byte). However only the 1st character of the string is used - the remainder is ignored. A null string ("") always matches a null char(string) and returns True. Otherwise, if either parameter is empty ("") then False is returned

See also: **FillString**

Function - IsValidVariant

Declare: Public Declare Function **IsValidVariant** Lib "mslib145.dll" (ByRef v As Variant) As Boolean

Purpose: Makes rudimentary checks on a Variant to ensure it is valid and is not corrupted. Accepts a Variant pointer (ByRef), checks the pointer is non-null and that invalid VT_* field bit-values are not set. These checks are not conclusive.

Function - Join

Declare: Public Declare Function **Join** Lib "mslib145" (ByRef V As Variant, _
Optional ByVal Separator As String = vbNullString, _
Optional ByVal RemoveEmptyItems As Boolean = False) As String

Purpose: An emulation of the VB6 Join() function for use with VB5 and other compatible-languages. Join takes an array of strings in either String Array, Variant or explicit Variant array format and concatenates them into a single string optionally including a separator string. Unlike VB6 this version permits empty strings to be ignored and separator characters for empty strings to be therefore omitted.

Notes: Although declared as taking type Variant() the VB will pass both String and Variant String array types correctly to this function, as well as ambiguous Variant types.

If the Separator value is omitted then strings will be separated by a space.
If the Separator value is passed as an empty string; ""; then strings will not be separated and will be contiguous in the resulting string
A non-empty Separator value will be inserted in between each string

Normally where empty strings are encountered their presence is revealed by superfluous separator characters. (See example below) You can suppress this behaviour by setting the optional "RemoveEmptyItems" value to True. This will cause only non-empty strings to be returned wrapped in the given separators. This parameter is an enhancement over the VB6 function may otherwise be ignored and left unset.

This function does not presently have a limit value as with the VB6 equivalent.

You may alias or omit the declaration for this function when used in VB6 since Join is a native function in this case.

The output from other VBToolbox functions which return a String array held in a non-array Variant may be passed into Join

Join makes formatting CSV and similar data easy and is similar to the PHP function called "implode"

Example:

```
Option Base 0
Dim a(3) as String     ' Example using explicit String array
a(0)="Zero"
a(1)="One"
a(2)=""
a(3)="Three"
Dim s As String
s=Join(a,",")          ' Don't remove blank/empty strings
Debug.Print "s=[";s;"]"
```

or:

```
Option Base 0
Dim a(3) as Variant    ' Example using explicit Variant String array
a(0)="Zero"            ' Assign String data
a(1)="One"
a(2)=""                ' Create an empty array item
a(3)="Three"
Dim s As String
s=Join(a,",")          ' Don't remove blank/empty strings
Debug.Print "s=[";s;"]"
```

Result:

```
Zero,One,,Three
```

Continued...

Function - Join (Continued)

Example:

```
Dim v as Variant       ' Ambiguous non-array Variant declaration
v=StrSplit("The-quick-brown-fox","-") ' Returns a Variant()
s=Join(v," ")
Debug.Print "s=[";s;"]"
```

Result:

```
The quick brown fox
```

Example:

```
' Example using automatic intermediate Variant
' StrSplit() breaks-up via a full match on the string mask
' Split and remove empty strings
Debug.Print Join(Tokenise("$1$2$3a$$$5$$", "$"), ",", True)
```

Result:

```
1,2,3a,5
```

Example:

```
' Example using automatic intermediate Variant
' Tokenise() breaks-up via *any* character token in the mask
' Tokenise and remove empty strings
Debug.Print Join(Tokenise("$1$2$3a$$$5$$", "$a2"), ",", True)
```

Result:

```
1,3,5
```

Example:

```
' Create a CRLF-formatted text list from ListFiles() output
Debug.Print Join(ListFiles("c:\MyFiles\", "*.txt"), vbCrLf)
```

Result:

```
A.txt
Basic.txt
Reminder.txt
Visual Basic.txt
```

See also: **Filter, PrintR, StrSplit, Tokenise**

Sub - Lower

Declare: Public Declare Sub **Lower** Lib "mslib145.dll" (ByVal S As String)

Purpose: Converts a string to lower-case. The parameter string is changed

---

Function - LowerStr

Declare: Public Declare Function **LowerStr** Lib "mslib145.dll" (ByVal S As String) As String

Purpose: Converts a string to lower-case. The parameter string is unaffected
You MUST use the return parameter. This function may NOT be called as if it were a subroutine. To convert a String in-situ use Sub Lower()
This routine is intended to be a high-speed replacement for LCase$

Notes: UpperStr uses the "C" case-conversion routines for both ANSI and Unicode.
Visual BASIC "Quirks" or "Stooges" are not intentionally reproduced.

See also: **Upper, UpperStr**

Sub - MatchBrackets

Declare: Public Declare Function **MatchBrackets** Lib "mslib145.dll" (ByVal s As String, _
ByVal LH As String, _
ByVal RH As String) As Long

Purpose: Matches any pair of bracket characters (or other matchable characters) indicating a mismatch where one exists between pairings.

Notes: Any pair of characters may be used.
Only the first character of any string is used. WhiteSpace counts as a character
The function is intended for use testing complex expressions such as RTF or mathematical-expressions for validity.

The function returns the following values:

0	String is empty or the pair of characters are evenly matched
>0	Number of excess LH brackets
<0	Number of excess RH brackets

Example:
```
Debug.Print MatchBrackets("{{}{}{{{}}{}{{}{{{}","{","}")
```

Result:
```
5 ' Five excess { brackets
```

See also: **BracketStr**

Sub - MidCharStr

Declare: Public Declare Function **MidCharStr** Lib "mslib145.dll" (ByVal s As String,
ByVal Ch As String, _
Optional ByVal ReverseSearch As Boolean = False) As String

Purpose: Searches a string and splits it as with Mid$ except the split is made where a given character matches rather than by absolute position. Using **MidCharStr** can save a lot of external arithmetic.

You may search either forwards from the start of the string or backwards from the end of the string. The default is to search from the start of the string.

Only the first character of the "searched-for" string is used.

The function is case-significant

The original string is not changed

Example:
```
Debug.Print MidCharStr("The quick fox quickly jumps","q")
```

Result:
```
"quick brown fox quickly jumps"
```

Example:
```
Debug.Print MidCharStr("The quick fox quickly jumps","q",True)
```

Result:
```
"quickly jumps"
```

See also: **MidStrStr**

Sub - MidStrStr

Declare: Public Declare Function **MidStrStr** Lib "mslib145.dll" (ByVal s As String,
ByVal ch As String, _
Optional ByVal ReverseSearch As Boolean = False) As String

Purpose: Searches a string and splits it as with Mid$ except the split is made where a given substring matches rather than by absolute position. Using **MidStrStr** can save a lot of external arithmetic.

You may search either forwards from the start of the string or backwards from the end of the string. The default is to search from the start of the string.

The function is case-significant

The original string is not changed

Example:
```
Debug.Print MidStrStr("The quick fox quickly jumps","quick")
```

Result:
```
"quick brown fox quickly jumps"
```

Example:
```
Debug.Print MidStrStr("The quick fox quickly jumps","quick",True)
```

Result:
```
"quickly jumps"
```

See also: **MidCharStr**

Function - PrintR

Declare: Public Declare Function **PrintR** Lib "mslib145.dll" (ByRef V As Variant, _
Optional ByVal ShowEmpty As Boolean = False) As Long

Purpose: An emulation of the PHP print_r() function. This prints out a variable in human-readable format. The function is provided as an aid to debugging VBToolbox programs. Output will be sent to an open console window.

Notes: Currently only one-dimensional String(), String Variant, Variant(), numeric arrays and non-array numeric variables are handled. A Variant input is accepted which means other functions which return a Variant type may be chained together with the result output to PrintR

PrintR returns the count of displayed elements where an array is shown, otherwise the number of bytes in the given variable is returned.

Example:
```
Debug.Print PrintR(StrSplit("1,2,3,4", ","))
```

Result:
```
ByVal:Array     Variant->String(4)
(
        [0] => "1"
        [1] => "2"
        [2] => "3"
        [3] => "4"
)
```

See also: **Filter, StrSplit, Tokenise**

Function - QSort

Declare: Public Declare Function **QSort** Lib "mslib145.dll" (ByRef V As Variant, _
Optional ByVal ReverseSort As Boolean = False, _
Optional ByVal IgnoreCase As Boolean = False) As Variant

Purpose: An implementation of the "C" QuickSort applying to Visual BASIC String() arrays or to Variant() arrays of Strings. The quicksort is an efficient means of sorting large arrays of data.

Notes: This implementation is flexible and may sort in both forward and reverse direction as well as being able to perform a case-insensitive or case-sensitive sort based on "C" case-comparison rules.

The function accepts a Variant type input which permits the output from other functions such as StrSplit or Tokenise to provide input. A Variant String array is returned, the output of which may be chained into other functions which accept a Variant as input such as PrintR

Both input and output from QSort may be "piped-lined" or "queued" from other VBToolbox Variant functions such as Join, StrSplit, Tokenise, PrintR etc.

Example:

```
Debug.Print PrintR(Join(QSort(StrSplit("1,two,3,four,5,six,7", ",",))))
```

Returns:

```
ByVal: VT_0x0008: Variant->String BSTR => "1 3 5 7 four six two"
```

See also: **Filter, PrintR, QSortStr, QSortVal, StrSplit, Tokenise**

Function - QSortStr

Declare: Public Declare Function **QSortStr** Lib "mslib145.dll" (ByRef s() As String, _
Optional ByVal ReverseSort As Boolean = False, _
Optional ByVal IgnoreCase As Boolean = False, _
Optional ByVal Reserved As Boolean = False) As Boolean

Purpose: An implementation of the "C" QuickSort applying to Visual BASIC String() arrays only. The quicksort is an efficient means of sorting large arrays of data. QSortStr avoids "casting" the variable to/from a Variant type.

Due to Automation-interface limitations QSortStr is not available in the TLB export version only the ANSI/Declare version. Use QSort() with the TLB version.

Notes: This implementation is flexible and may sort in both forward and reverse direction as well as being able to perform a case-insensitive or case-sensitive sort based on "C" case-comparison rules.

Use QSort instead to sort Variant() arrays of strings as well as String() arrays

The "Reserved" parameter should not be changed from it's default setting

To sort numeric values use QSortVal

See also: **Filter, PrintR, QSort, QSortVal, StrSplit, Tokenise**

Function - QSortVal

Declare: Public Declare Function **QSortVal** Lib "mslib145.dll" (ByRef v As Variant, _
Optional ByVal ReverseSort As Boolean = False) As Variant

Purpose: An implementation of the "C" QuickSort applying to Visual BASIC numeric arrays only. The quicksort is an efficient means of sorting large arrays of data.

QSortVal currently supports only homogeneous arrays of either Integer, Long or Double. You **cannot** mix data-types within an Variant array you wish to sort. A Variant array must be of a consistent type. The same function will sort any of the distinct types of data either declared directly or passed in a Variant array.

Notes: This implementation is flexible and may sort in both forward and reverse direction.

It is not possible to sort diverse Variant data-types within the same array properly or logically. A Visual BASIC Variant array will let you store any type of data within the same array (mixed or heterogeneous data types). For example, VB will let you store Double, String, Boolean and Integer within the same array. Whilst it may be plausible to cast and sort between integer data types, keeping the sort routine fast and efficient precludes translating between inconsistent types in the same array.

QSortVal will automatically detect Variant array types from the first non-empty array element and will check that the rest of the array is consistent with that type.

Where mixed Variant array elements are encountered (other than NULL and EMPTY) an Empty, non-arrayed Variant will be returned. You can test this return using the VB IsArray() function.

A new, sorted Variant array is returned and the parameter array is also sorted

Since the function returns a Variant you cannot directly assign the return to an array of the original data-type unless that is a also a Variant.. You can, however, Call the function and ignore the return value to sort the original array in-situ.

You may use this one function to sort the following array data types:

Byte (as Integer -Static)	Dim x(10) As Byte
Byte (as Integer - Dynamic)	Dim x() As Byte
Integer (Static)	Dim x(10) As Integer
Integer (Dynamic)	Dim x() As Integer
Long (Static)	Dim x(10) As Long
Long (Dynamic)	Dim x() As Long
Double (Static)	Dim x(10) As Double
Double (Dynamic)	Dim x() As Double
Date (as Double - Static)	Dim x(10) As Date
Date (as Double - Dynamic)	Dim x() As Date
Variant() (Static) (with above types)	Dim x(10) as Variant
Variant() (Dynamic) (with above types)	Dim x() As Variant

Unsupported types at present are Boolean, Single, Decimal, Currency, Object and user-defined. These may be added in a future release.

Example:

```
Dim v as Variant(10)          ' Sorted, Fixed Variant
Dim rv as Variant             ' Return Variant
Dim i As Integer              ' Loop counter
For i = LBound(v) To UBound(v)
       v(i) = Round(CDbl(Random(99, 100000) / 33), 2)
Next i
rv = QSortVal(v, False)       ' Sort in normal-order

Call PrintR(rv, True)         ' Show any empty elements/extra info
```

Result:

```
ByRef: VT_0x600c: Array of Variant->Variant(11)
(
        [0]     Double => 17.8800
        [1]     Double => 93.7600
        [2]     Double => 175.8800
        [3]     Double => 304.8500
        [4]     Double => 364.8800
        [5]     Double => 512.9100
        [6]     Double => 708.4800
        [7]     Double => 744.3300
        [8]     Double => 820.0300
        [9]     Double => 855.8800
        [10]    Double => 892.6400
)
```

Example:

```
Dim ba(5) As Byte      ' Define a Byte-array
Dim i as Integer
For i = LBound(ba) To UBound(ba)
       ba(i) = cByte(Random(0, 100))
Next
Call QSortVal(ba, False)      ' Sort the original array
```

Result:

```
ByRef: VT_0x6011: Array of Variant->Byte(6)
(
        [0]     Byte => 0
        [1]     Byte => 41
        [2]     Byte => 85
        [3]     Byte => 72
        [4]     Byte => 38
        [5]     Byte => 80
)
```

See also: **Filter, PrintR, QSort, QSortStr, StrSplit, Tokenise**

Function - Replace

Declare:

```
#ifdef VB5          'Defined in the default module
Public Declare Function Replace Lib "mslib145.dll" (ByVal s As String, ByVal
        SearchedFor As String, ByVal Replacement As String) As String
#endif
```

Purpose: Search for and replace instances of one string by another one
(searches for needle string in haystack s)
The replacement string may be empty
It is permissible for the entire contents of the string to be replaced resulting in an empty or null string being returned.

Examples:

```
Replace("quick brown fox","brown","red")     ' Returns "quick red fox"
Replace("quick brown fox","brown"," ")       ' Returns "quickredfox"
Replace("aHaealalao","a","")          ' Returns "Hello"
Replace("aaaaaaaaaa","a","")          ' Empty or "NULL" string
Replace("Hello","","H")               ' (NULL return)
Replace("","q","r")           ' Empty "haystack" (NULL return)
Replace("","","r")            ' No search parameters (NULL return)
Replace("01234",0,9)          ' Valid - VB casts numbers as strings
                              ' Returns "91234" (0 is passed as "0")
Replace("012",012,A)          ' Valid - VB casts numbers as string-
'equivalents but take care since leading zeroes will be ignored except
'for 0 in the Immediate pane. This is a VB5 problem usual in the
'immediate window.
'Returns "0A" (012 is passed as "12") in the immediate window
```

Notes: **Replace()** is an intrinsic (built-in) function for Visual BASIC 6 - ensure that **#Const VB5 =True** is set correctly. or commented-out as needed.

Always use valid string values even if VB will "cast" numeric values to string equivalents. The use of non-string values may yield unexpected results.

See also: **ReplaceChar**

Function - ReplaceChar

Declare:

```
Public Declare Function ReplaceChar Lib "mslib145.dll" (ByVal s As String, ByVal
        SearchedFor As String, ByVal Replacement As String) As String
```

Purpose: Character replacement only. Replaces each occurrence of the "SearchedFor" character in "s" with the first character of "Replacement". Only the first character of "SearchedFor" and "Replacement" are used - remaining characters are ignored.

Examples:

```
Debug.Print ReplaceChar("Hello there","e","E")
```

Prints out "HEllo thErE"

Notes: For Use With VB 4.0 and VB5.0 only. Redundant in VB 6.0 and above as the Replace() function is "built-in". The returned string is exactly the same length as the supplied string parameter. In order to simplify calling the function, a string is accepted as the 2nd and 3rd parameters rather than a char (Byte). However only the 1st character of the parameter string is used - the remainder is ignored.

See also: **Replace**

Function - ReverseWords

Declare: Public Declare Function **ReverseWords** Lib "mslib145.dll" _
(ByVal s As String) As String

Purpose: Reverses individual words in a plain ANSI (ASCII) string

Notes: Reverses only individual words. Does not reverse the entire text

The delimiter characters are:
Space, Tab, Carriage return, (0x0d) and Linefeed (0x0a)
Where CRLF pairs are met they are expected to be in the order 0x0d,0x0a

Example:

```
Debug.Print "The quick brown fox jumps over-the lazy dog"
```

Result:

```
ehT kciuq nworb xof depmuj eht-revo yzal god
```

See also: **StrRev**

Function - SliceLeft

Declare: Public Declare Function **SliceLeft** Lib "mslib145.dll" (ByRef s As String, _
ByVal RemoveBytes As Long) As Boolean

Purpose: Permits the reduction of a string by removal of "N" characters from the left-hand side of the string without necessitating intermediate variables in VB. This is a vital issue when handling extremely large strings in VB of, say, up to several hundred megabytes in size.

The usual means of extracting would be to use Mid$ as a function and return into either the same or a new string variable. However, in either case this requires the creation of DOUBLE the memory allocation, even if this is for a short-duration.

SliceLeft avoids impacting on the VB string space and achieves this by allocating memory directly using the Windows API and using the "C" memcpy() function to copy before deleting the original string and reallocating. For these reasons there is no string return and the original string is re-allocated via the function body parameter.

SliceLeft is "binary" string compatible and handles only ANSI strings

Example:

```
Dim s as String
s=ReadFileFromString("c:\100mb.bin") ' 100mb file - 104857600 bytes
s=SliceLeft(s,10)      ' Slice 10 characters from the left
Debug.Print Len(s)
```

Result:

```
104857590
```

Function - StrCSpan

Declare: Public Declare Function **StrCSpan** Lib "mslib145.dll" (ByVal s As String, _
ByVal MaskString As String) As Long

Purpose: Returns a base 1 value indicating the location of the first character of the set defined by "MaskString". If no character is found then 0 is returned.

Notes: This is similar in functionality to the "C" function strcspn() but StrCSpan gives proper indication where the character-class is not found (0).

It is important to know that the return value is "base 1" indexed. That is the first character is 1 not 0. Results for empty parameters are as follows.

strcspan("","") => 0
strcspan("","abc") => 0
strcspan("abc","") => 0
strcspan("abc","abc") => 1

Example:

```
Debug.Print StrCSpan("Hello, World",".;:,")
```

Result:

```
6
```

Function - StripLStr

Declare Public Declare Function **StripLStr** Lib "mslib145.dll" (ByVal s As String, ByVal Mask As String) As String

Purpose Takes a string and truncates it by removing **any** characters on the Left-hand side that match the template pattern of the 2nd parameter. Since any characters in the 2nd parameter are allowed to match they can be specified in any order.

Example: `StripLStr(" Hello world"," leH")`

Result: `"o world"`

Function - StripL

Declare: Public Declare Function **StripL** Lib "mslib145.dll" (ByVal s As String, ByVal Mask As String) As String

Purpose: Duplicates the **LTrim**() function in VB but will trim any character not just spaces.

Function - StripRStr

Declare Public Declare Function **StripRStr** Lib "mslib145.dll" (ByVal s As String, ByVal Mask As String) As String

Purpose Takes a string and truncates it by removing **any** characters on the Right-hand side that match the template pattern of the 2nd parameter. Since any characters in the 2nd parameter are allowed to match they can be specified in any order.

Example: `StripRStr("  Hello world"," dlrw")`

Returns: `"Hello wo"`

Function - StripR

Declare: Public Declare Function **StripR** Lib "mslib145.dll" (ByVal s As String, ByVal Mask As String) As String

Purpose: Duplicates the RTrim() function in VB but can trim any character not just spaces.

Function - StrRev

Declare: Public Declare Function **StrRev** Lib "mslib145.dll" (ByVal S As String) As String

Purpose: In-situ. Reverses a VB string

See also: **ReverseWords**

Function - StrSplit

Declare: Public Declare Function **StrSplit** Lib "mslib145.dll" (ByVal ArgString As String, _
Optional ByVal Delimiter As String = "", _
Optional ByVal ArrayBase As Integer = 0, _
Optional ByVal IgnoreCase As Boolean = False) As Variant

Purpose: StrSplit is similar to Tokenise and the VB6 Split() function. It takes a string and transforms it into an array by "slicing" it where matches are found on the Token string.

Notes: StrSplit differs from Tokenise in that StrSplit slices the string only where a full-match is made on the entire Delimiter parameter String; whereas Tokenise matches any single character in the Delimiter String.

IF the Delimiter string is not found an array with a single-element starting at "ArrayBase" containing the original string is returned.

You may specify the array base using the "Base" parameter. The default is 0
If you wish you can make a case-insignificant comparison of the Token by setting "IgnoreCase" to True.

The resulting String array is returned within a non-array Variant object which may be indexed in the usual way using UBound(), LBound() etc.

StrSplit is very similar to the PHP function "explode"

Parameters: **Delimiter:** Optional for ANSI Declare version. Mandatory for TLB version
A String character used to identify substring boundaries.
If omitted, the space character (" ") is assumed to be the delimiter.
If delimiter is a zero-length string, (vbNullString) a single-element array containing the entire expression string is returned.
vbNullString or "" may be supplied as the parameter to the TLB version

ArrayBase: Optional. Determines the array base of the returned array. The default is zero. You may change this to match the Option Base value.

IgnoreCase: Optional. Whether or not to ignore case when matching the Delimiter string. The default value is False but you may modify the Declare statement to change this behaviour.

Example:

```
Dim v as Variant
v = StrSplit("$1$2$3a$$$5$$", "$")
For i = LBound(v) To UBound(v)
       Debug.Print "v("; i; ")=["; vt(i); "]"
Next
```

Result:

```
v( 0 )=[]
v( 1 )=[1]
v( 2 )=[2]
v( 3 )=[3a]
v( 4 )=[]
v( 5 )=[]
v( 6 )=[5]
v( 7 )=[]
v( 8 )=[]
```

See also: **Join, PrintR, Tokenise, VariantToArray**

Function - SwapStr

Declare: Public Declare Sub **SwapStr** Lib "mslib145.dll" (ByRef a As String, _
ByRef b As String)

Purpose: Swaps two VB strings by swapping the string pointers instead of swapping the actual string-data

Notes: Under some circumstances it may be faster and more efficient to use SwapStr. e.g. Sorting routines and array transposition

---

Function - Tokenise

Declare: Public Declare Function **Tokenise** Lib "mslib145.dll" (ByVal s As String, _
Optional ByVal Delimiters As String = "", _
Optional ByVal ArrayBase As Integer = 0, _
Optional ByVal IgnoreCase As Boolean = False) As Variant

Purpose: Single-character tokenisation. Breaks up a linear String into a two-dimensional String-array of tokens by means of a specified String list of individual character delimiters. This is similar to operations performed using the "C" strtok() function. This is a common and often complex problem to implement. The string will be divided at the point at which each (if any) of the individual character delimiters are found within the argument string

Notes: This function is similar to the native VB6 function called "Split()" (see StrSplit)

The string array is returned in a Variant which can be checked using the standard functions IsEmpty(), Ubound(), Lbound() etc. before processing.
One or more characters must be specified in the token-separator list. The string will be broken-up each time one of the individual character tokens specified is found.

The Delimiters parameter is mandatory with the TLB/Unicode version
If the Delimiters parameter is NULL or empty "" then a space character is assumed
If no token-separators are found, the original string will be returned in an array containing only one element. The base index of the array can be set using the optional parameter "Base". The default array-base is zero.

Tokenise() is used to implement the Console function, **GetArgs**()
Visual BASIC is responsible for maintaining the returned Variant. No subsequent deletion is performed by Tokenise(). Internally, Tokenise makes use of SAFEARRAY data types which hold BSTR strings.
Join can be used to reassemble a tokenised string
Tokenise is similar to the PHP function "explode"

Unicode: Note that, this function can accept and return Unicode strings and should be "Unicode-safe". This is a requirement due to VB not performing an automatic "cast" to ANSI on any returned object.

Example:

```
Dim s as String
Dim v as Variant
Dim i as Integer
s="The quick, brown fox jumped over the lazy dog"
v=Tokenise(s," ")      ' Divide at each space character
If IsArray(v) Then
        For i=LBound(v) to Ubound(v)
                Debug.Print v(i)
        Next
End If
```

Result:

```
The
quick,
brown
fox
jumped
over
the
lazy
dog
```

See also: **ArgFound, ArgVal, GetArgs, GetCGIArgs, Join, PrintR, StrSplit**

Sub - Upper

Declare: Public Declare Sub **Upper** Lib "mslib145.dll" (ByVal S As String)

Purpose: Converts a string to upper-case. The parameter string is changed

---

Function - UpperStr

Declare: Public Declare Function **UpperStr** Lib "mslib145.dll" (ByVal S As String) As String

Purpose: Converts a string to upper-case. The parameter string is unaffected
You MUST use the return parameter. This function may NOT be called as if it were a subroutine. To convert a String in-situ use Sub Upper()
This routine is intended to be a high-speed replacement for UCase$

Notes: LowerStr uses the "C" case-conversion routines for both ANSI and Unicode.
Visual BASIC "Quirks" or "Stooges" are not intentionally reproduced.

See also: **Lower, LowerStr**

Function - WordWrap

Declare: Public Declare Function **WordWrap** Lib "mslib145.dll" (ByVal s As String, _
ByVal Width As Long, ByRef NewLen As Long) As String

Purpose: Wraps a string, if possible at the nearest whitespace or hyphen division.

Notes: The new length of the string is returned in the "NewLen" parameter
If the string contains over-length words or suitable break-points cannot be found then the words will be broken at the specified boundary.
This is a simple text-wrap and it makes no attempt to implement font kerning

---

Function - WildcardMatch

Declare: Public Declare Function **WildcardMatch** Lib "mslib145.dll" (ByVal S As String, ByVal Wildcard As String, Optional IgnoreCase As Boolean = False) As Boolean

Purpose: Simple character-based text matching. This does NOT offer any kind of regex-like functionality.

Notes: Characters are matched in more or less the same was as when searching for MS-DOS filenames.

A "*" character will stand for any group of characters.
A "?" will stand for any single character.

The remainder will match on a literal basis. You can specify "IgnoreCase" to make a case-insensitive comparison. The default is a case-sensitive comparison.

Function - WordCount

Declare: Public Declare Function **WordCount** Lib "mslib145.dll" (ByVal s As String) As Long

Purpose: Performs a word-count on a String..

Notes: Words are defined here as any alpha-numeric group of characters. Thus any other character than A..Z, a..z or 0..9 counts as white-space. Words may be hyphenated.

Hyphenated words are counted as a single word but only where the hyphen occurs between valid words and with no intervening space; thus "one-word" is a single word but "one – word" counts as two words due to intervening spaces. Prefixed and postfixed hyphens (minus characters) are not valid.

Example:

```
Dim s as String
s="   the -quick brown- fox jumps over the lazy-dog  "
Debug.Print WordCount(s)
```

Result:

```
8
```

Function - WordList

Declare: Public Declare Function **WordList** Lib "mslib145.dll" (ByVal s As String, _
Optional ByVal ArrayBase As Integer = 0) As Variant

Purpose: Performs an array of words from a given string as would be processed by WordCount()

Notes: Words are defined here as any alpha-numeric group of characters. Thus any other character than A..Z, a..z or 0..9 counts as white-space. Words may be hyphenated.

Hyphenated words are counted as a single word but only where the hyphen occurs between valid words and with no intervening space; thus "one-word" is a single word but "one – word" counts as two words due to intervening spaces. Prefixed and postfixed hyphens (minus characters) are not valid.

Example:

```
Dim s as String
s="   the -quick brown- fox jumps over the lazy-dog  "
Debug.Print PrintR(WordList(s))
```

Result:

```
ByVal: VT_0x2008: Array of Variant->String(8)
(
        [0] => "the"
        [1] => "quick"
        [2] => "brown"
        [3] => "fox"
        [4] => "jumps"
        [5] => "over"
        [6] => "the"
        [7] => "lazy-dog"
)
```

Arithmetic and Number Functions

This includes various useful math, math-related and random-number generation functions

Function - Ceil

Declare: Public Declare Function **Ceil** Lib "mslib145" (ByVal d As Double) As Double

Purpose: Exposes the "C" ceil() function. Rounds a number up to the next highest integer

See also: **Floor, Round**

Function - DecimalToRoman

Declare: Public Declare Function **DecimalToRoman** Lib "mslib145.dll" (_
ByVal i As Integer) As String

Purpose: Converts an Indo-Arabic (decimal) value to classical Roman notation using the letters IVXLCDM. Numbers may be converted back using **RomanToDecimal**.

Notes: The function can handle values ranging from 0 to 10,000
Values can be confirmed or tested using the Google Calculator
Signed negative values are accepted
The result is returned in upper-case format

Example:
```
Debug.Print DecimalToRoman(9999)
```

Result:
```
MMMMMMMMMCMXCIX
```

See also: **RomanToDecimal, RomanDigitToDecimal**

Function - Floor

Declare: Public Declare Function **Floor** Lib "mslib145" (ByVal d As Double) As Double

Purpose: Exposes the "C" floor() function. Rounds a number down to the next lowest integer

See also: **Ceil, Round**

Function - FMod

Declare: Public Declare Function **FMod** Lib "mslib145.dll" (ByVal d As Double, ByVal v _
As Double) As Double

Purpose: Exposes the "C" fmod() function. The Visual BASIC Mod function rounds values to Integer range before performing math which limits the range to "Integer Max" or 2,147,483,647. The "C" fmod() function operates within the Double data-type range which permits a higher range to be evaluated..

See also: **Round**

Function - Gcd

Declare: Public Declare Function **Gcd** Lib "mslib145.dll" (ByVal a As Long, ByVal b As Long) As Long

Purpose: Calculate the Greatest Common Denominator (GCD) of 2 numbers

Example: `Gcd(325,20)  returns the value 5`

Function - Max

Declare: Public Declare Function **Max** Lib "mslib145.dll" (ByVal a As Integer, ByVal b As Integer) As Integer

Purpose: DLL integer version of Max()

Notes: Both Max() and Min are commonly implemented with VB code but it is much faster for repetitive calls if called from a DLL.

Function - Min

Declare: Public Declare Function **Min** Lib "mslib145.dll" (ByVal a As Integer, ByVal b As Integer) As Integer

Purpose: DLL integer version of Min()

Notes: Both Max() and Min are commonly implemented with VB code but it is much faster for repetitive calls if called from a DLL.

Function - PiStr

Declare: * Function removed - v1.22 - 11 September 2009 *

Notes: To be migrated to a separate PI.DLL

Function - RomanToDecimal

Declare: Public Declare Function **RomanToDecimal** Lib "mslib145.dll" (_
ByVal s As String) As Integer

Purpose: Converts a string holding a numeric value in classic Roman notation to an Indo-Arabic (decimal) value. Numbers may be converted in the opposite direction using **DecimalToRoman**

Notes: The function can handle values ranging from Roman "i" (1) to "MMMMMMMMMM" (10,000).
Upper, lower or mixed-case input is acceptable
Spaces and tab characters within the string are permitted
Values can be confirmed or tested using the Google Calculator
High-level Roman "bar" characters for multiples of 1,000 are not handled

Example:
```
Debug.Print RomanToDecimal("MMMMMMMMMCMXCIX")
```

Result:
```
9999
```

Example:
```
' Test code
For i = 1 To 9999
    If RomanToDecimal(DecimalToRoman(i)) <> i Then
        Debug.Print "failed at "; i
            Debug.Print "DecimalToRoman(i)="; DecimalToRoman(i)
            Debug.Print "["; RomanToDecimal(DecimalToRoman(i));"]"
            End       ' Halt at error
     End If
     Debug.Print i, DecimalToRoman(i)
Next
Debug.Print "* Pass *"
```

Result:
```
"* Pass *"
```

See also: **RomanToDecimal, RomanDigitToDecimal**

Function - RomanDigitToDecimal

Declare: Public Declare Function **RomanDigitToDecimal** Lib "mslib145.dll" (_
ByVal c As Byte) As Integer

Purpose: Converts a Byte value holding a Classic Roman numeric digit which is one of "IVXLCM" to a decimal value

Notes: To call using VB String values use the Asc() function
Case is not significant during conversion

Example:
```
Debug.Print RomanBitToDigital(Asc("V"))
```

Result:
```
5
```

Function - MTRandomise

Declare: Public Declare Sub **MTRandomise** Lib "mslib145.dll" (Optional ByVal Seed _ As Integer)

Purpose: Implements the "*mt19937*" Mersenne Twister Programmable Random Number Generator (PRNG). This function initialises the RNG. If not called directly with a seed-value then the first call to any MT* function will initialise it.

See also: Mersenne Twister home page at: http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

---

Function - MTRnd

Declare: Public Declare Function **MTRnd** Lib "mslib145.dll" (ByVal Low as Long, _ ByVal High as Long) as Long

Purpose: Uses the Mersenne Twister PRNG to generate a random-Long Integer between "Low" and "High" (inclusive)

Function - MTRndDouble

Declare: Public Declare Function **MTRndDouble** Lib "mslib145.dll" (ByVal Low as Long, _ ByVal High as Long) as Long

Purpose: Uses the Mersenne Twister PRNG to generate a random-Double between "Low" and "High" (inclusive)

Function - MTRandomStr

Declare: Public Declare Function **MTRandomStr** Lib "mslib145.dll" (ByVal length As _ Long, Optional ByVal Style As Integer = RandomStringMixedCase) As String

Purpose: Generates a sequence of random values using the Mersenne Twister random-number generator according to the given Style parameter. The Style parameter values are the same as for RandomStr().

See also: RandomStr

Function - Random

Declare: Public Declare Function **Random** Lib "mslib145.dll" (ByVal Lo As Integer, ByVal Hi As Integer) As Integer

Purpose: Returns a random number in the range 0.65535 between the bounds of parameters Hi and Lo, inclusive.

Notes: You can initialise the VBToolbox (CRT) random number generator by calling Randomise()

Function - Randomise

Declare: Public Declare Function **Randomise** Lib "mslib145.dll" (Optional ByVal Seed As Integer) As Integer

Purpose: Sets/resets the "C" random-number generator. The seed value is optional. If no seed is specified then the clock time will be used to set it.

Notes: If using encryption functions derived from **Random()** then you need to be sure to consistently call Randomise() with the same seed *before* each individual encryption/decryption phase

---

Function - RandomStr

Declare: Public Declare Function **RandomStr** Lib "mslib145.dll" (ByVal Length As Long, ByVal Style As Integer) As String

Purpose: Creates a string of the given length, allocating memory and filling with characters according to the specified style. Five different styles of fill are provided:

```
Public Const RandomStringMixedCase = 0
Public Const RandomStringLower = 1
Public Const RandomStringUpper = 2
Public Const RandomStringNumeric = 3
Public Const RandomStringAll = 4
Public Const RandomStringBinary = 5
Public Const RandomStringHex = 6
Public Const RandomStringBinaryString = 7
Public Const RandomStringBinaryStringBias0 = 8
Public Const RandomStringBinaryStringBias1 = 9
Public Const RandomStringHexLower = 6
```

Notes: Useful for generating random filenames or random padding text for encryption.

RandomStringBinary creates a string with full binary range 0x00..0xff
RandomStringBinaryString creates a text-string with "1" and "0" characters
RandomStringBinaryStringBias0 creates a string as with RandomStringBinaryString but biased by 25% towards producing "0" characters (75%-25% ratio). This is useful for testing RLE and other data-compression.
RandomStringBinaryStringBias1 creates a string as with RandomStringBinaryString but biased by 25% towards producing "1" characters (75%-25% ratio). This is useful for testing RLE and other data-compression.

Function - Integral

Declare: Public Declare Function **Integral** Lib "mslib145.dll" (ByVal d As Double) As Double

Purpose: Returns the integer (integral) part of a double variable.
This function safely exposes the C/C++ modf() function

Function - Fraction

Declare: Public Declare Function **Fraction** Lib "mslib145.dll" (ByVal d As Double) As Double

Purpose: Returns the fraction or "decimal" part of a double variable.
This function safely exposes the C/C++ modf() function

Function - Round

Declare: Public Declare Function **Round** Lib "mslib145.dll" (ByVal d As Double,
Optional ByVal Places As Integer) As Double

Purpose: Rounds a VB double variable up to "n" places

Example:

```
Dim x as Double
x=123.456789
Round(x)        ' -> 123
Round(x,1)      ' -> 123.5
Round(x,2)      ' -> 123.46
Round(x,3)      ' -> 123.457
Round(x,4)      ' -> 123.4568
```

Notes: This function may be *aliased* if required to avoid name-conflicts with later versions of Visual BASIC or other languages.

Example alias:

Public Declare Function **VBRound** Lib "mslib145.dll" Alias "**Round**"
(ByVal d As Double, Optional ByVal Places As Integer) As Double

Date and Time Functions

Function - UKToISODate

Declare: Public Declare Function **UKToISODate** Lib "mslib145.dll" (ByVal DDsMMsYYYY As String) As String

Purpose: Converts a string in *full* UK DD-MM-YYYY date format to ISO format (YYYYMMDD). Improperly formatted inputs return "00000000" (8 zeroes).

Examples:

```
UKToISODate("10-01-1969")
returns the string  "19690110"

UKToISODate("08+04-1968Hi!")
returns  "19680408" (R/H-excess is OK and is ignored)

UKtoISODate("08/04/68")
returns "00000000"  (2-digit years are not allowed)

UKtoISODate("8/4/68")
returns "00000000"  (single digits are not allowed)
```

Notes: The string must be a minimum of 10 characters but any excess over that is ignored The string must be formatted exactly in dd/mm/yyyy format. 2-digit years are not allowed. Any separator character can be used (they are ignored).

No date-specific checks are made for the validity of the date held in the string. ISO (International Standards Organisation) format time is also commonly known as "Military Format" time in the US.

This function is the inverse of **ISOToUKDate**

See also: **UKShortToISODate** for non-Y2K format date conversion.
PHPDate, **PHPDateNow**

Function - ISOToUKDate

Declare: Public Declare Function **ISOToUKDate** Lib "mslib145.dll" (ByVal ISODate As String) As String

Purpose: Convert a date string in pure ISO format to a formatted UK-date string
Input is in the format "yyyymmdd" and output is "dd/mm/yyyy"

Notes: Checks are made on the length of the string and that only numeric characters are passed to the routine No further date checks are made. It is up to the calling routine to check if the date given and returned is actually a valid date. (e.g. not 49th December 2049.

Date inputs which are not exactly 8 characters or which contain non numeric characters cause a string "00/00/0000" to be returned.

ISO (International Standards Organisation) format time is also known as "Military Format" time in the US.

This function is the inverse of **UKToISODate**

Function - UKShortToISODate

Declare: Public Declare Function **UKShortToISODate** Lib "mslib145.dll" (ByVal DDsMMsYY As String) As String

Purpose: Convert a "short" format UK date string into and ISO-date format string

Examples:
```
UKShortToISODate("08/04/68")  returns the string  "19680408"
UKShortToISODate("08/04/50")  returns the string  "20500408"
UKShortToISODate("08/04/49")  returns the string  "20490408"
UKShortToISODate("08/04/68Hi!")  returns the string  "19680408"
UKShortToISODate("08+04*68")  returns the string  "19680408"
```

Notes: Automatic "Century rollover" is performed as follows…
For dates over YY=50 dates are assumed to be 2050 onwards
For dates under YY=50 dates are assumed to be 1949 or lower

Due to this inaccurate representation of years use of 2-digit dates is **not** recommended. Any code written will have an accurate "shelf-life" of less than 50 years and could eventually return incorrect dates.

The supplied string must be a minimum of 10 characters, any excess is ignored. Primitive checks are made on the formatting of the string, separator characters such as "/" or "-" are ignored and no checks are made that the date is valid.

ISO (International Standards Organisation) format time is also known as "Military Format" time in the US.

See also: **UKToISODate**

Function - IsLeapYear

Declare: Public Declare Function **IsLeapYear** Lib "mslib145.dll" (ByVal Year As Integer) As Boolean

Purpose: Returns True or False depending whether the year parameter is a leap year

Example:
```
IsLeapYear(2000)  returns "True"
```

Notes: Accurate to within the next century (after 2000 AD)

Function - NumOrd

Declare: Public Declare Function **NumOrd** Lib "mslib145.dll" (ByVal N As Integer) As String

Purpose: Returns a 2-character text postfix for a given number

Example:
```
Debug.Print "1";NumOrd(1),"2";NumOrd(2),"3";NumOrd(3)
```

Returns:
```
"1st           2nd           3rd"
```

Notes: Accurate to within the next century

Function - PHPDate

Declares: Public Declare Function **PHPDate** Lib "mslib145.dll" (ByVal d As Date, Optional ByVal s As String) As String

Purpose: Emulates the highly-flexible PHP date() function for any Visual BASIC date in the PHP/**Unix** Epoch of **1st January 1970** to the **5th of February 2036** (inclusive). For the full list of marker characters see the table below. See also PHPDateNow() A properly formatted double such as 39619.8489467593 may be used as input in place of a Visual BASIC date.

This function does not cover the full Visual BASIC date-range.

Errors: For dates outside the acceptable Unix epoch the string "ERR" is returned

Notes: **Important** - When VB creates a date object it does not adjust for local daylight-savings-time (DST) within the date-object itself. If you want an accurate representation of the current time with DST adjustment you must use PHPDateNow instead.

For example the following code will NOT work if DST is in-force:
Debug.Print PHPDate(Now) 'Will print -1 hr out if DST +1 is in force

Swatch metric time is not implemented PHPDate("B") will return "0".
Milliseconds are not yet implemented since the granularity of the date() function is less than about 100 milliseconds.
Static Date object variable passing is not yet supported but may be so in the future

Example:
```
Debug.Print PHPDate(Now,"r")
```

Returns:
```
"Sat, 28 Aug 2010 19:36:00 +0000"
```

Example:
```
Debug.Print PHPDate(25569)    ' Start of the Unix epoch
```

Returns:
```
"Thu, 1st January 1970 00:00:00"
```

Function - PHPDateNow

Declares: Public Declare Function **PHPDateNow** Lib "mslib145.dll" (Optional ByVal s As String) As String

Purpose: Emulates the highly-flexible PHP date() function for the current system date providing it lies within the PHP/Unix Epoch of 1st January 1970 to the 5th of February 2036 (inclusive).

For the full list of marker characters see the table applying to PHPDate
See also PHPDate

Notes: Correctly adjusts for Daylight Savings Time (DST)
For dates outside the acceptable Unix epoch the string "ERR" is returned

---

Function - DSTAdjust

Declares: Public Declare Function **DSTAdjust** Lib "mslib145.dll" (ByVal d As Date)
As Double

Purpose: Adjusts for local Daylight Savings Time (DST or "Summer Time")
When Visual BASIC creates a date variable no accounting is included for DST and the function "Now()" returns a value devoid of DST adjustment. This means that if you use the VB "Now()" function to get the current time and then use it with PHPDate() it will display the wrong time. This time will represent the unadjusted time (UTC).

Commonly +1 hour or, in rare cases, +2 hours might be added for local DST. Normally you should use PHPDateNow() for the current system date and time, However, If you wish to use PHPDate with a date/time value relevant to the current system-locale then you can adjust for DST using the DSTAdjust() function.

Example: Where the current system-time shows 01:48:53 and a +1 hour DST adjustment is in operation:

```
Debug.Print PHPDate(Now,"r")
Prints out "Fri, 20 Jun 2008 00:48:53 +0000"

Debug.Print PHPDate(DSTAdjust(Now),"r")
Prints out "Fri, 20 Jun 2008 01:48:53 +0000"
```

Notes: A +1 hour adjustment would result in 0.0417 being added to the current "decimal time value". This represents an adjustment of +1hr x 1/1440th of a second.

No adjustment is made for localised time zones by this function

Table - PHPDate and - PHPDateNow Token Characters

Char	Name	Description	Example
\	Escape flag	Escape a literal character	"\T\i\m\e\ \i\s \n\o\w H:i:s"
A	AM/PM	Upper-case Ante Meridian marker	"AM"
a	am/pm	Lower-case Ante Meridian marker	"am"
B	Swatch Metric Time	Not implemented	(returns "0")
C	C/C++ asctime()	Print out the "C"/C++-style asctime() string. The \n (LF) character is not included and is thus < 26 chars long. Implemented as the macro - "D M d H:i:s Y" Not-PHP/non-standard	"Sat Jun 21 22:46:24 2008"
c	ISO 8601 date	Implemented as the macro - "Y-m-d\TH:i:sO"	"2004-02-12T15:19:21+00:00"
D	Day name 3	Day name - length 3 characters max	"Mon"
d	Day number	Day of the month - with leading zeroes	"01"
e	Time Zone	Time Zone identifier string This is always local to the PC and not part of any supplied date variable	"GMT Standard Time"
F	Month name (full)	Full month name	"January"
g	Hour - H	12-hour format hour value with no leading zeroes	"9" (pm)
H	Hour - HH	24-hour format hour value with leading zeroes	"21" (pm)
h	Hour - HH	12-hour format hour value with leading zeroes	"09" (pm)
I	DST Query	1 if Daylight Savings Time (DST) enabled - 0 if not This is always local to the PC and not part of any supplied date variable	"0" (not in force)
i	Minutes	Minutes formatted with leading zero	"09"
j	Month Day	Day of the month with NO leading zeroes - 1..31 maximum	"1"
L	Leap Year	Leap Year Query. 1 if True, 0 if not Valid only for dates in the Unix epoch	"1"
l	Full Day Name	Full day name	"Monday"
M	Month Name 3	3 digit month name prefix	"Jan"
mm	Month Number	2-digit month number with zero prefix	"01"
O	GMT Diff	Difference from GMT in hours with +/- sign. This is always local to the PC and not part of any supplied date variable	+0100
P	GMT Diff Colon	Difference from GMT in hours with +/- sign and separating colon This is always local to the PC and not part of any supplied date variable	+01:00

r	ISO822 RFC	RFC formatted date string - implemented as a macro - "D, d M Y H:i:s O"	"Fri, 20 Jun 2008 02:10:21" +0000"
S	Ordinal Day	Ordinal day of the month	"st"
s	Seconds	Second value with leading zero	"01"
t	Month Days	Days of the month	"30"
U	Unix Epoch	Seconds elapsed since the start of the Unix Epoch on 1st January 1970	"1213928007"
ww	Weekday	Weekday number starting with 0 for Sunday and ending at 6 for Saturday	"0"
Y	Full Year	Full, 4-digit year	"2008"
y	Short Year	Two-digit year value	"08"
Z	Time Zone	Time Zone offset in seconds from GMT-0 This is always local to the PC and not part of any supplied date variable	"0"
z	Year Day	Day of the Year - starting at zero	"0"

Notes: PHPDate() and PHPDateNow() both implement all date tags of the equivalent PHP function except:

"B" - (Swatch metric time)
"u" - (milliseconds)

Additional non-PHP (non-standard) tags are given as:

"C" - (capital C) - "C"/C++ asctime() format

Note that "literal" characters may be retained by "escaping" with a backslash character the same as with the PHP version of date() - e.g. "\T\o\d\a\y \i\s l" will print - "Today is Monday"

There should be little need to embed literals within a date-format string unless there is a practical need to embed literal characters deep within a complex date format as with the "C" macro flag.

Function - VBDateStr

Declares: Public Declare Function **VBDateStr** Lib "mslib145.dll" (ByVal d As Date) As String

Purpose: Returns the internal representation of a Visual BASIC date object. This is returned as a string representation of a double. The integral part of this number represents the number of days since the start of the VB epoch at midnight on 30th December 1899 to the end of December 1999.

Notes: The date is stored in decimal format. The fractional component represents part of a single day in nanoseconds. The last 3 digits of this represent the time in milliseconds.

The exact double value of 0.0 represents "00:00:00 30 hrs on December 1899". Negative numbers are permitted which extend the range backwards in time. Thus, the earliest date and time in the Date range, 00:00:00 1 January 100 maps to the negative double value -657434.0. The highest permitted date and time in the Date range, is "23:59:59 31 December 9999" (10,000AD -1 second) which is represented by the Double value 2958465.99998843.

Example:
```
Debug.Print VBDateStr(Now)
```

Returns:
```
"40383.905428240738" (24/07/2010 21:43:49)
```

Function - VBDateMsecs

Declares: Public Declare Function **VBDateMsecs** Lib "mslib145.dll" (ByVal d As Date) As Integer

Purpose: Returns the number of milliseconds stored in a Visual BASIC date object

Notes: *Experimental only.* This value is normally inaccessible. not exposed by VB and is apparently updated only every 1 second

Example:
```
Debug.Print VBDateMsecs(now)
```

Returns:
```
189
```

Function - VBDateToCTime

Declares: Public Declare Function **VBDateToCTime** Lib "mslib145.dll" (ByVal d As Date) As Long

Purpose: Converts a Visual BASIC date variable and returns it as a long formatted as a "C" style **time_t** variable.

Notes: If the date supplied is outside the Unix epoch or otherwise invalid then -1 is returned to indicate the error.

Example:
```
Dim d as Date
d=#6/20/2008 10:12:01 PM#
Debug.Print "VBDateToCTime(d)="VBDateToCTime(d)
```

Returns:
```
"VBDateToCTime(d)= 1213992721 "
```

Function - DateToHex

Declares: Public Declare Function **DateToHex** Lib "mslib145.dll" (ByVal d As Date) As String

Purpose: Converts a Visual BASIC date variable into "storage" hex-format. This is a precise byte-representation of the variable's memory image and is 100% precise with no rounding effects incurred during storage and retrieval.

Notes: The VB Date variable is a "C"-style Double value.
The inverse function HexToDate() returns the original Date or Double value

Example:
```
Debug.Print DateToHex(#6/20/2008 10:12:01 PM#)
```

Returns:
```
62DFB1997D58E340
```

See also: **HexToDate, MKD, CVD**

Function - HexToDate

Declares: Public Declare Function **HexToDate** Lib "mslib145.dll" (ByVal s As String)
As Double

or:

Public Declare Function **HexToDate** Lib "mslib145.dll" (ByVal s As String)
As Date

Purpose: Converts a Visual BASIC date variable which has been converted into "storage" hex-format back into either a VB Double or Date variable. This conversion method is an exact byte-representation of the variable's memory image and is 100% precise with no rounding effects incurred during storage and retrieval. This makes it suitable for cross platform storage such as in SQL: databases where accuracy could be lost in storing as a converted form of Double.

Notes: It is important to note that the VB Date variable is a "C"-style Double value.
Declaring as a Date rather than Double merely causes VB to interpret the date differently. The inverse function, DateToHex(), is used to convert the value to hex format
Either declaration may be used. If the Double return version is chosen then Cdate() must be used in order for VB to interpret the value as a Date
The hex string can only be retrieved by HexToDate() on the same O/S platform and CPU-architecture since it will be byte-order and implementation dependent.

Example:
```
Debug.Print HexToDate(DateToHex(#6/20/2008 10:12:01 PM#))
```

Returns:
```
Either 39619.9250115741 or "6/20/2008 10:12:01 PM"
depending on the declare used.
```

See also: **DateToHex, MKD, CVD**

Legacy BASIC Conversion Functions

Visual BASIC 5.0 has the following standard numeric data types which correspond with many "C"/C++ compiler standard numeric types as char, int, long, float and double. The function names are acronyms for "make" and "convert" - i.e. "make integer string" is Mki and convert to integer Cvi.

Byte (not implemented)	1 byte 0 to 255 (unsigned)
Integer	2 bytes -32,768 to 32,767 ("C" short integer)
Long (long integer)	4 bytes -2,147,483,648& to 2,147,483,647&
Single (single-precision floating-point)	4 bytes -3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
Double (double-precision floating-point)	8 bytes -1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values

Microsoft Visual C++ has basic data types in the range which either match or are a very close match to those in VB ...

short	Two bytes - -32,768 to 32,767
int	Two or Four bytes (System dependent)
long	four bytes, -2,147,483,648 to 2,147,483,647
float	four bytes - 3.4E +/- 38 (7 digits)
double	eight bytes - 1.7E +/- 308 (15 digits)

Functions - Mki and Cvi (Integer)

Declares: Public Declare Function **Mki** Lib "mslib145.dll" (ByVal I As Integer) As String
Public Declare Function **Cvi** Lib "mslib145.dll" (ByVal Str As String) As Integer

Purpose: Convert a VB integer value (equivalent to a two-byte "C" int from -32,768 to 32,767) to a two-byte binary-string representation

Example:
```
Debug.Print Cvi(Mki(32000))
```

Returns:
```
"32000"
```

Functions - Mkl and Cvl (Long)

Declares: Public Declare Function **Mkl** Lib "mslib145.dll" (ByVal L As Long) As String
Public Declare Function **Cvl** Lib "mslib145.dll" (ByVal Str As String) As Long

Purpose: Convert a VB long value (equivalent to a two-byte "C" long in the range -2,147,483,648 to 2,147,483,647) to a four-byte binary-string representation

Example:
```
Debug.Print Cvl(Mkl(1818181818))
```

Returns:
```
"1818181818"
```

Functions - Mkf and Cvf (Float)

Declares: Public Declare Function **Mkf** Lib "mslib145.dll" (ByVal F As Single) As String
Public Declare Function **Cvf** Lib "mslib145.dll" (ByVal Str As String) As Single

Purpose: Convert a VB long value (equivalent to a four-byte "C" float in the range 3.4E +/- 38 (7 digits)) to a four-byte floating-point binary-string representation.

Example:
```
Debug.Print Cvf(Mkf(66666.66))
```

Comment: Note that rounding can be expected from the fraction part of the number

---

Functions - Mkd and Cvd (Double)

Declares: Public Declare Function **Mkd** Lib "mslib145.dll" (ByVal D As Double) As String
Public Declare Function **Cvd** Lib "mslib145.dll" (ByVal Str As String) As Double

Purpose: Convert a VB long value (equivalent to eight-byte "C" double in the range 1.7E +/- 308 (15 digits)) to an eight-byte floating-point binary-string representation.

This is useful for storing VB Date objects as these are stored internally as Double values.

Example:
```
Debug.Print Cvd(Mkd(66666666666.666))
```

Returns:
```
66666666666.666
```

See also: **DateToHex, HexToDate**

Data Encoding Functions

Various functions which transform or convert data.

Function - BaseConv

Declare: Public Declare Function **BaseConv** Lib "mslib145.dll" (ByVal x As Integer, ByVal Base As Integer) As String

Purpose: Convert any VB integer value (Byte, Integer or Long) to a string representation in any base from 2 to 16 inclusive.

Example:
```
BaseConv(451,3)              returns the string  "121201"
BaseConv(451,2)              returns the string  "111000011"
BaseConv(451,16)             returns the string  "1C3"
```

Comment: Note that the VB "Long" data-type is an unsigned value, hence conversion of values in the upper-range where the sign-bit is set will trigger an overflow (Error 6). Example: BaseConv(HexToLong("FFFFFFFF")) will trigger this error. Use BaseConvDouble for values outside 0x7FFFFFFF

Function - BaseConvDouble

Declare: Public Declare Function **BaseConvDouble** Lib "mslib145.dll" (_
ByVal x As Double, ByVal Base As Integer) As String

Purpose: Convert any Double VB integer value to a string representation in any base from 2 to 16 inclusive for values in the range (0 to 4,294,967,295) Although a Double can hold far higher values accuracy of conversion of Double integers values is limited. Maximum input range is therefore set at 0xFFFFFFFFFFFFF (4503599627370495)

Example:
```
BaseConvDouble(68719476735,8)        returns the string  "777777777777"
BaseConvDouble(1099511627775,12)     returns the string  "159114277313"
BaseConvDouble(728121033505,16)      returns the string  "A987654321"
```

Comment: Note that the VB "Long" data-type is an unsigned value, hence conversion of values in the upper-range where the sign-bit is set will trigger an overflow (Error 6).

As the Visual BASIC Long data-type is a signed value in the range -2,147,483,648 to 2,147,483,647 you should use **BaseConvDouble**() in preference to **BaseConv**() for 4-byte Unsigned Long values in the range (0 to 4,294,967,295)

BaseConvDouble is intended primarily to handle the range of a 32-bit/4-byte "unsigned long" (0xFFFFFFFF) but is accurate up to 52 bits or 0xFFFFFFFFFFFFF (4.50359963 × 10^15) at base 2.

VB Wrapper - Base$()

Declare: Public Function **Base$**(x As Variant, BaseVal As Integer)

Purpose: Wrapper function for **BaseConv**

Converts a VB Integer value (Byte, Integer or Long) to a string in any base value from 2 to 16

Example:

```
For i=2 to 16
       Debug.Print i, Base$(32767,i)
Next
```

Result:

```
 2            111111111111111
 3            1122221121
 4            13333333
 5            2022032
 6            411411
 7            164350
 8            77777
 9            48847
 10           32767
 11           22689
 12           16B67
 13           11BB7
 14           BD27
 15           9A97
 16           7FFF
```

Comment: Returns from Base$() do not need to be converted using VBStr()
You can call BaseConv() directly if you wish but the wrapper function provides some degree of safety and type-protection.

You can use Base$() for numbers outside of the range of a Visual BASIC signed-long (-2,147,483,648 to 2,147,483,647 (7FFFFFFF)) which is the maximum value BaseConv() can handle within LongToHex()

Function - Bin8, Bin16, Bin32

Declares: Public Declare Function **Bin8** Lib "mslib145.dll" (ByVal i As Byte) As String
Public Declare Function **Bin16** Lib "mslib145.dll" (ByVal i As Integer) As String
Public Declare Function **Bin32** Lib "mslib145.dll" (ByVal i As Long) As String

Purpose: Number-to-binary string conversion

Example: Bin16(234) gives the resulting string "0101011100000000"

Notes: You must use the correct version for the data type you are calling with or unpredictable things can happen.
The inverse function for all of these is BinToDec()

Note also that some VB functions such as "Val()" always return a "Double" which is not supported. In such cases you need to convert to the correct type using…

CByte() Converts to Byte data type
CInt() Converts to Integer data type
CLng() Converts to Long data type

Use Bin32(CLng(Val("&h00FFFFFFFF"))) rather than Bin32(Val("&h00FFFFFFFF"))

VB Wrapper - Bin$()

Declares: Public Function **Bin$(**x As Variant)

Include File: MSLIB145.BAS

Purpose: Wrapper function for Bin8(), Bin16() and Bin32() conversion functions
Bin$() will automatically detect Byte, Integer or Long data types and call the correct number-to-binary string functions for you.

Notes: In case of problems "Debug.Print" trace lines are included in the function.
Note that some VB functions always return a "Double" which needs to be converted to the correct data type for the conversion functions.

Function - BinToDec

Declares: Public Declare Function **BinToDec** Lib "mslib145.dll" (ByVal s As String) As Long

Purpose: Convert a binary string produced by Bin$() etc. al. to a decimal numeric value

Example:
```
BintoDec("1101")       Returns  13
BintoDec("&b1101")     Returns  13
```

Notes: Maximum range of 32 bits. (0.. 2147483647 for signed values)
Supplied text can be any length from 1 to 32 characters in length
The case-insignificant prefix "&b" is accepted - e.g. "&b1001110"
Non-bit value characters cause evaluation to terminate and the number to be evaluated for "n" characters to that point - e.g. "1111abc0100" = 15 decimal
Excess text over 32 characters is ignored.

There is only one function, which returns a VB Long - you can use CInt() or CByte() to convert to other numeric ranges.

There is no VB Double equivalent - use Cdbl() to convert

Function - DecToBin

Declares: Public Declare Function **DecToBin** Lib "mslib145.dll" (ByVal i As Long, _
Optional ByVal StripLeadingZeroes As Boolean = False) As String

Purpose: A convenient wrapper for the Bin8, Bin16 and Bin32 conversion functions

Notes: The function automatically-detects which of the 3 binary conversion functions to call. Additionally, leading zeroes may be omitted from the returned string. Bin8..Bin32 usually return fixed-length strings which include leading zeroes.

See also: **Bin8, Bin16, Bin32, BinToDec**

Function - HiByte

Declares: Public Declare Function **HiByte** Lib "mslib145" (ByVal i As Integer) As Byte

Purpose: Returns the topmost Byte (8-bits) of a Visual BASIC 16-bit Integer (Word)

Function - HiWord

Declares: Public Declare Function **HiWord** Lib "mslib145" (ByVal l As Long) As Integer

Purpose: Returns the topmost Integer or Word (16-bits) of a Visual BASIC 32-bit Long

---

Function - LoByte

Declares: Public Declare Function **LoByte** Lib "mslib145" (ByVal i As Integer) As Byte

Purpose: Returns the lowermost Byte (8-bits) of a Visual BASIC 16-bit Integer (Word)

Function - LoWord

Declares: Public Declare Function **LoWord** Lib "mslib145" (ByVal l As Long) As Integer

Purpose: Returns the lowermost Integer or Word (16-bits) of a Visual BASIC 32-bit Long

---

Functions - Bit-Rotation - RotlByte, RotrByte, RotlInt, RotrInt, RotlLong, RotrLong

Declares:

Public Declare Function **RotlByte** Lib "mslib145" (ByVal b As Byte, ByVal Bits As Byte) As Byte
Public Declare Function **RotrByte** Lib "mslib145" (ByVal b As Byte, ByVal Bits As Byte) As Byte
Public Declare Function **RotlInt** Lib "mslib145" (ByVal i As Integer, ByVal Bits As Byte) As Integer
Public Declare Function **RotrInt** Lib "mslib145" (ByVal i As Integer, ByVal Bits As Byte) As Integer
Public Declare Function **RotlLong** Lib "mslib145" (ByVal l As Long, ByVal Bits As Byte) As Long
Public Declare Function **RotrLong** Lib "mslib145" (ByVal l As Long, ByVal Bits As Byte) As Long

Description: Offers logical bit-rotation which can be implemented in "C" using the ">>" and "<<" bitshift operators. Function nomenclature is *Sh[Left|Right][Type]*

Notes: Note that this is a logical bitshift without carry and not an arithmetic one.
For more information see the following Wikipedia page:
http://en.wikipedia.org/wiki/Circular_shift

Logical bit-rotation is useful for encryption and graphics image processing

There is no Double bit-rotate

Example:

```
Dim ByteVal as Byte
ByteVal = &HCC         ' Set a bit-pattern (0b11001100) to rotate
For i = 0 To 10
       Debug.Print "RotLByte(0b"; Bin$(ByteVal); ",";
       Debug.Print Format(i, "000"); ")=0b"; Bin$(RotlByte(ByteVal, i))
Next
```

Result:

```
RotLByte(0b11001100, 000)=0b11001100  ' Identity @ 0
RotLByte(0b11001100, 001)=0b10011001
RotLByte(0b11001100, 002)=0b00110011
RotLByte(0b11001100, 003)=0b01100110
RotLByte(0b11001100, 004)=0b11001100
RotLByte(0b11001100, 005)=0b10011001
RotLByte(0b11001100, 006)=0b00110011
RotLByte(0b11001100, 007)=0b01100110
RotLByte(0b11001100, 008)=0b11001100  ' Identity @ 8
RotLByte(0b11001100, 009)=0b10011001
RotLByte(0b11001100, 010)=0b00110011
```

See also: **RotateByte, RotateInt, RotateLong**

Functions - Bit-Shifting - ShlByte, ShrByte, ShlInt, ShrInt, ShlLong, ShrLong

Declares:

Public Declare Function **ShlByte** Lib "mslib145" (ByVal b As Byte, ByVal Bits As Byte) As Byte
Public Declare Function **ShrByte** Lib "mslib145" (ByVal b As Byte, ByVal Bits As Byte) As Byte
Public Declare Function **ShlInt** Lib "mslib145" (ByVal i As Integer, ByVal Bits As Byte) As Integer
Public Declare Function **ShrInt** Lib "mslib145" (ByVal i As Integer, ByVal Bits As Byte) As Integer
Public Declare Function **ShlLong** Lib "mslib145" (ByVal l As Long, ByVal Bits As Byte) As Long
Public Declare Function **ShrLong** Lib "mslib145" (ByVal l As Long, ByVal Bits As Byte) As Long

Description: Offers logical bit-shifting which is implemented in "C" using the ">>" and "<<" bitshift operators. Function nomenclature is *Sh[Left|Right][Type]*

Bits are shifted either to the left or right by "N" bits as specified with the void filled with zero values. Identity occurs every zero or modulus BITWIDTH bits shifted.

Notes: The Bits parameter is masked to BITWIDTH-1 (e.g. 0..7 for Byte) and repetition will occur with values higher than BITWIDTH

Even if the Bits parameter were not masked to BITWIDTH, identity still occurs with compiled DLL code but at a much larger and less-intuitive interval. Such behaviour is compiler-dependent.

This function offers a logical rather than arithmetic bit-shift. Sign bits are not preserved. For more information see the following Wikipedia page:
http://en.wikipedia.org/wiki/Logical_shift

Logical bit-shifting is useful for encryption and graphics image processing

There is no Double bit-shift

Example:

```
ByteVal = &HFF                ' 0b11111111
For i = 0 To 10
       Debug.Print "Shlbyte("; ByteVal; ","; i; ")=";
       Debug.Print Bin$(ShlByte(ByteVal, i))
Next
```

Result:

```
Shlbyte( 255 ,  0 )=11111111  ' Identity @ 0
Shlbyte( 255 ,  1 )=11111110
Shlbyte( 255 ,  2 )=11111100
Shlbyte( 255 ,  3 )=11111000
Shlbyte( 255 ,  4 )=11110000
Shlbyte( 255 ,  5 )=11100000
Shlbyte( 255 ,  6 )=11000000
Shlbyte( 255 ,  7 )=10000000
Shlbyte( 255 ,  8 )=11111111  ' Identity @ 8
Shlbyte( 255 ,  9 )=11111110
Shlbyte( 255 , 10 )=11111100
```

See also: RotateByte, RotateInt, RotateLong

Function - RotateByte

Declare: Public Declare Function **RotateByte** Lib "mslib145.dll" (ByRef b As Byte) As Byte

Purpose: Rotates or "inverts" the high and low nibbles of a single byte.

Notes: This nibbles are swapped from left to right i.e. AB -> BA, for example.
0X1F -> 0xF1 (hexadecimal). Values are passed ByRef (BYTE*)

Note that the return may be interpreted in some cases as signed value when the top (leftmost) bit is set

The function may be called repeated to re-invert to the original value

Example:
```
Debug.Print RotateByte("C9")
```

Result:
```
"9C"
```

See also: Bit-Shifting functions, Bit-Rotating functions

Function - RotateInt

Declare: Public Declare Function **RotateInt** Lib "mslib145.dll" (ByRef i As Integer) As Integer

Purpose: Rotates or "inverts" the high and low nibbles of a single two-byte integer.

Notes: Note that on some systems an integer may be defined as a four-byte value

Bits are rotated in "little to big endian" order i.e.
ABCD -> DCBA - where each letter represents a single byte value
For example 0xF1E2 -> 0x2E1F (hexadecimal)

Values are passed ByRef (WORD*)

The return may be interpreted in some cases as signed value when the top (leftmost) bit is set

The function may be called repeated to re-invert to the original value

Example:
```
Debug.Print RotateInt("2E74")
```

Result:
```
"47E2"
```

See also: Bit-Shifting functions, Bit-Rotating functions

Function - RotateLong

Declare: Public Declare Function **RotateLong** Lib "mslib145.dll" (ByRef l As Long) As Long

Purpose: Rotates or "inverts" the high and low nibbles of a four-byte long integer

Notes: Note that on some systems an integer may be defined as a four-byte value

Bits are rotated in "little to big endian" order i.e.
ABCDEFGH -> HGFEDCBA - where each letter represents a single byte value, for example 0xF1E2D3C -> 0xC3D2E1F (hexadecimal). Values are passed ByRef (long*)

The return may be interpreted in some cases as signed value when the top (leftmost) bit is set

The function may be called repeated to re-invert to the original value

Example:
```
Debug.Print RotateLong("1A236F77")
```

Result:
```
"77F632A1"
```

Example:
```
Debug.Print RotateLong(RotateLong("1A236F77"))
```

Result:
```
"1A236F77"
```

See also: Bit-Shifting functions, Bit-Rotating functions

Function - StrToHex

Declare: Public Declare Function **StrToHex** Lib "mslib145.dll" (ByVal s As String, _
ByVal Length As Long, Optional ByVal WrapWidth As Integer) As String

Purpose: Converts a binary or text-string into a hexadecimal-coded string. Each character is represented by its two-byte "hex" pair. The resulting string may, optionally, be line-wrapped by CRLF pairs after "WrapWidth" byte-pairs. Note that this is byte-pairs not characters.

Example:
```
Debug.Print StrToHex("Hello World!",12)
```

Prints out: "48656C6C6F20576F726C6421"

Comment: The returned string is always **twice** as long as the supplied ANSI string. If line-wrapping is enabled then this length is increased by 2 bytes for each unit of WrapWidth. The length of the returned string is limited only by the available memory on the PC. A quick ASCII reference is that "A" = 65 (0x41), "a" = 97 (0x61)

Since StrToHex() may be used to encode "binary" strings containing embedded NULL characters such as those produced by EncryptString() this version requires the length of the passed string to be specified.

See also: StringToHex, HexToStr

Function - StringToHex

Declare: Public Declare Function StringToHex Lib "mslib145.dll" (ByVal s As String) As String

Purpose: An alternate to StrToHex() where the encoded string can be guaranteed NOT to have embedded NULL (0x00) characters within it. Where there is a likelihood of embedded NULL characters you should use StrToHex() instead and keep track of/pass the string length value.

See also: StrToHex, HexToStr

Function - HexToStr

Declare: Public Declare Function **HexToStr** Lib "mslib145.dll" (ByVal s As String)
As String

Purpose: Converts a "hex-coded" string, as produced by StrToHex into its original form.

Example:
```
Debug.Print HexToStr("48656C6C6F20576F726C6421")
```

Result:
```
"Hello World!"
```

Comment: The inverse of StrToHex(). The length of the returned string is always **half** the original size

See also: StrToHex, StringToHex

Function - HexToChar

Declare: Public Declare Function **HexToChar** Lib "mslib145.dll" (ByVal HexPair As String) As Byte

Purpose: Returns the value (char/Byte in the range 0..255) of a 2-byte text string in hexadecimal format. This is the inverse of CharToHex.
Only valid hex characters in the range "0".."F" are allowed.

The value may be supplied either as raw hex ("FF"), VB Style hex, (&HFF) or "C" style hex (0xFF).

Example:
```
HexToChar("A2")  'returns the Byte value   162
```

Function - HexToInt

Declare: Public Declare Function **HexToInt** Lib "mslib145.dll" (ByVal Str As String) As Integer

Purpose: Convert a string value of *up-to* 2 valid hex-bytes (4 characters/nibbles) into a signed integer value in the range -32,768 to 32,767. This is not returned "unsigned" due to intrinsic the signed nature of the VB integer data-type.

The value may be supplied either as raw hex ("FF"), VB Style hex, (&HFF) or "C" style hex (0xFF).

For unsigned values within the range of an 16-bit unsigned int (WORD) use **HexToLong**. Only valid hex characters in the range "0".."F" are allowed. Any other value returns zero.

Function - HexToLong

Declare: Public Declare Function **HexToLong** Lib "mslib145.dll" (ByVal Str As String) As Double

Purpose: Convert a string value of *up-to* 4 valid hex-bytes (8 characters/nibbles) to a VB interpretation of an unsigned long. This is emulated by returning as a double to ensure the value held by the "C" DLL is preserved. Values are returned in the range 0x0 to 0xFFFFFFFF (0 to 4294967295).

The value may be supplied either as raw hex ("FF"), VB Style hex, (&HFF) or "C" style hex (0xFF).

String values longer than 8 characters return zero.
Only valid hex characters in the range "0".."F" are allowed.

Function - HexToDouble

Declare: Public Declare Function HexToDouble Lib "mslib145.dll" (ByVal Str As String) As Double

Purpose: Convert a 32-bit hexadecimal string value of *up-to* 8 valid hex-bytes (16 characters/ nibbles) into a VB representation held as a Double. Only valid hex characters in the range "0".."F" are allowed.

Notes: Due to the limitations of useful data-types in Visual BASIC. precise accuracy is lost as representation changes to floating-point values much larger than 0x2 FFFF FFFF FFFF (844424930131967). Values of 0x3FFF FFFF FFFF and larger are represented in approximated "scientific" notation -
e.g. 1.12589990684262E+15 for 0x3FFF FFFF FFFF.

Reliability is good for numbers up to 48-bits (0xFFFFFFFFFFFF)

The value may be supplied either as raw hex ("FF"), VB Style hex, (&HFF) or "C" style hex (0xFF).

Calculations are performed internally using 64-bit integers before the value is converted and returned as a Double. Due to differing floating-point representations it is difficult to compare and check 100% during development and testing. However the function has been checked using several methods and is certainly accurate up to 0x2 FFF FFFF FFFF (844424930131967). Values outside this range return as follows...

Hex value	Result	Check/Comments
0xFFFF FFFF FFFF	281474976710655	(24-bits)
0x1 FFFF FFFF FFFF	562949953421311	(25-bits)
0x2 FFFF FFFF FFFF	844424930131967	(26-bits)
0x3 FFFF FFFF FFFF	1.13E+015	(Google = 1.12589991 × 10^15)
0x4 FFFF FFFF FFFF	1.41E+015	(Google = 1.40737488 × 10^15)
0x9 FFFF FFFF FFFF	2.81E+015	(Google = 2.81474977 × 10^15)
0xA FFFF FFFF FFFF	3.10E+015	(Google = 3.09622474 × 10^15)
0xF FFFF FFFF FFFF	4.50E+015	(WinCalc) = 4503599627370495)
0xFF FFFF FFFF FFFF	7.21E+016	(Google 7.2057594 × 10^16)
0xFFF FFFF FFFF FFFF	1.15E+018	(Google 1.1529215 × 10^18)
0xFFFF FFFF FFFF FFFF	-1	(Google 1.84467441 × 10^19) (Error)

Function - CharToHex

Declare: Public Declare Function **CharToHex** Lib "mslib145.dll" (ByVal i As Byte) As String

Purpose: Takes an unsigned 1 byte character (Byte) and returns a numeric value in a 2-byte hex-character formatted-string in the range 0x00 to 0xff. ("00" to "FF") This is the inverse of HexToChar

Example:
```
CharToHex(65)     'Returns the string "41" (which is 65 in decimal)
```

Comment: The return value is always 2-bytes plus a terminating NULL character
Use IntToHex() or LongToHex() for values outside the 0..255 range

---

Function - IntToHex

Declare: Public Declare Function **IntToHex** Lib "mslib145.dll" (ByVal i As Integer) As String

Purpose: Takes a signed integer in the range 0..32767. Returns a 4-byte hex-formatted string integer value in the range 0x0000 to 0x7FFF ("0000" to "7FFF")

Example: `IntToHex(90)  'Returns the 4-byte string    "005A"`

Notes: Use CharToHex() for values in the range 0 to 255 or HexLong for larger values. Note that the built-in "Hex$()" can only handle values of signed long which makes it ineffective for processing IP address values of unsigned long where an Overflow (6) would result.

Function - LongToHex

Declare: Public Declare Function **LongToHex** Lib "mslib145.dll" (ByVal l As Double) As String

Purpose: Takes a 32-bit unsigned long in the range 0..4294967296 (0..0xFFFFFFFF). Returns an 8-byte hex-formatted string integer value in the range 0x0000 to 0x7FFF ("0000" to "7FFF")

Example: `IntToHex(90)  'Returns the 4-byte string    "005A"`

Notes: A double is used to pass from VB in order to ensure that the correct unsigned range is passed from VB without unnecessary conversion rather than a signed value max of 0x7FFFFFFF.

For numbers outside the range of a VB signed-long you can use the slower Base$() wrapper function or call Base() directly. Strings of ASCII and binary characters may be converted using StrToHex()

Function - BitPack

Declare: Public Declare Function **BitPack** Lib "mslib145.dll" (ByVal s As String, Optional ByVal Length As Long) As String

Purpose: Packs or compresses a numeric string into half of it's original length

The function will not compress or handle any other character than those specified. If optional parameter"Length" is specified then only this number of bytes will be packed and returned; the remainder will be ignored.

Notes: Valid characters are numeric digits 0 to 9, space, "+", "-", comma, and period only. Processing will halt and any resulting string will be truncated and returned at the first non-valid character. That is - `"0123456789 +-,."`)

Compression is 100% constant regardless of the number of bytes compressed and is not affected by random-entropy. Characters are compressed into a 4-bit encoding system which contains 2 characters per 8-bit output character.
Note that there is normally no useful compression-advantage in storing numbers as strings regardless of the effectiveness of any compression routine. For large values see BCD functions.

The returned string is a NULL-terminated string with characters being in the range of 17 to 255 decimal (0x11 to 0xFF) which must be processed using VBStr() before further direct use other than by **BitUnpack**() or another VBToolbox function.

The reduced length of a string containing an even number of characters is precisely half the original length (+ 1 byte for odd string lengths)

The string may be viewed using StrToHex() but cannot normally be printed out. BitPack() may be used to sequentially pack large quantities of numeric values for storage into database fields. Note that any field or storage used to hold the result must be capable of handling the full **unsigned char** range of 17 to 255 (0x11 to 0xFF) for each character and the result is, effectively a binary-string.

Can be used to store and compress very long numbers as text without resorting to linking-in a ZIP library. Was developed to store large PI values as a string.

Example:

```
Debug.Print Len(VBStr(BitPack(RandomStr(100,RandomStringNumeric ))))
Prints out "51" indicating a 49% reduction in length

Debug.Print BitUnPack(BitPack("0123456789"))
Prints out "0123456789" indicating cyclic consistency check pass OK

Debug.Print VBStr(StrToHex(BitPack("+,-. 0123456790abc")))
Prints out "BCDEF12345678A10"
```

Function - BitUnPack

Declare: Public Declare Function **BitUnPack** Lib "mslib145.dll" (ByVal s As String,
Optional ByVal BitPairs As Long) As String

Purpose: The inverse of **BitPack**. Unpacks (unzips) a numeric string which has been packed only by a call to **BitPack.** If optional parameter"Length" is specified then only this number of bit-pairs will be unpacked and returned; the remainder will be ignored.

Note that this will be bit-pairs not single-bytes.

Notes: Only strings which have been compressed by **BitPack** should be passed to this function otherwise the resulting string will contain unpredictable contents.

BitUnPack expands the length of a string by 100% (+/-1 byte for odd values)

The maximum string length which may be supplied is limited naturally by the output of **BitPack**. However if you manage to artificially construct **BitPack** binary strings then a string no longer than half the maximum VB string size should be supplied or a system fault may occur.

Example:

```
Debug.Print BitUnPack(BitPack("1234567890"))

Prints out "1234567890"       ' Cyclic test check pass OK
```

Function - DecodeString64

Declare: Public Declare Function **DecodeString64** Lib "mslib145.dll" _
(ByRef sInput As String, ByRef NewLength As Long) As String

Purpose: Decodes a string (binary or otherwise) that has been text-encoded in "base-64" format. By EncodeString64. Commonly used within encryption routines.

Notes: The return value is passed via the function body
A buffer is allocated internally and returned to Visual BASIC via the function body. VB takes care of deallocation and garbage-collection.

Since "binary" string containing embedded NULL (0x00) characters may be supplied to DecodeString64() it calculates the exact length of the allocated string and returns it via the "NewLength" parameter. You must not change the ByRef declaration for this variable.

If you wish to use nested "DecodeString64(EncodeString64(..." statements but don't wish to use the NewLength parameter supply a dummy variable: e.g.

Dim l as Long;
Debug.Print DecodeString64(EncodeString64("Hello World",l))

See also: **EncodeString64, RLEDecompress**

Function - EncodeString64

Declare: Public Declare Function **EncodeString64** Lib "mslib145.dll" (_
ByVal sInput As String, _
Optional ByVal WrapWidth As Integer = 0) As String

Purpose: Encodes a string (binary or otherwise) in text-encoded "base-64" format. Commonly used within email or encryption routines.

Notes: The return value is passed via the function body

A buffer is allocated internally and returned to Visual BASIC via the function body. VB takes care of deallocation and garbage-collection.

If WrapWidth is set to zero (default) then CRLF block-wrapping is disabled.

See also: **DecodeString64, RLECompress functions**

Function - EncryptString

Declare: Public Declare Function **EncryptString** Lib "mslib145.dll" (ByVal s As String, _
ByVal Length As Long, ByVal Password As String, _
Optional ByVal Salt As String = "", _
Optional ByVal ExtraSecure As Boolean = True) As Boolean

Purpose: Provides symmetric XOR-encryption of a text string using a password.

This does **not** offer a high-level of security as with PGP, AES or RSA, nevertheless it would not be trivial to crack the encrypted text without specialised software The longer the password the better as this is also hashed into the encryption routine along the length of the encoded block. To decrypt the string, call the function again with the encrypted string and the same password and stored length value and any optional Salt value given during encoding.

Notes: If parameter "ExtraSecure" is set to True then an "RC4-like" method of encryption is used which creates a randomised hash block from the password and block length. This block is then randomly rotated as the key is XORed into the data stream. This method although more secure should not be equated with AES or similar encryption. RC4-like ciphers have been easily cracked. If the block size value is changed with this value set then the ciphertext cannot be decrypted.

It is **vital** that the length parameter is passed when calling and be retained by the calling program. EncryptString creates a binary string which may randomly contain embedded NULL characters (0x00). C string handling will normally terminate when a NULL is encountered, thus precise string length handling is required to ensure the entire string is processed. You may choose to process less than the full length of the string. If, you make an error handling the length value and supply one which is too large then EncryptString may corrupt areas of memory which could make your program unstable or crash.

The password parameter is case-significant. The string is encoded "in-situ" and is not reallocated in memory; nor does the length of the string expand or contract. The length parameter may be shorter than the string length, in which case only "length" bytes of the string will be transcoded. If the length parameter given is longer than the allocated string then unpredictable results may happen including an application fault or system lock-up.

In-situ encryption is performed which does not change the length or memory allocation of the original string. You may call as either a Sub() and, in this case, ignore the return value or as a Function() and use the return value. Generally use as a Sub() with no return parameter can be expected to be more efficient and avoid extra memory-allocation overheads, particularly with very large strings (several megabytes in size).

You may optionally add a "Salt" value as a string. This is included into the hash algorithm. Salt values are commonly sent in plaintext with the ciphertext.

Encrypted strings would normally be stored in HEX or Base64 format and converted using **StrToHex** or **EncodeString64**

If you intend to compress encrypted blocks with say ZLIB then you should do this before calling EncryptString otherwise you will encounter no compression-gain.

Memory: EncryptString performs an in-situ encryption which does not allocate memory and does not change the length of the encoded data-block. See notes above about the need to carefully-control the length of the data-block being encoded or decoded.sss

Example:

```
Dim s as string
Dim r as string
Dim r As String: s = "Hello world"
Call EncryptString(s, Len(s), "secret",,True)
Debug.Print s
Call EncryptString(s, Len(s), "secret",,True)
Debug.Print s
```

Result:

```
j"%-W □u{=ø
Hello world
```

Notes: To properly handle the encrypted string you ***must*** keep a track of the full string length of the **original**, unencrypted string as the encrypted string may have embedded NULL characters which cause Len() to return a short value.

The returned string will always be the same length as the supplied plaintext one regardless of what value Len() shows for the resulting encoded string.

See also: RLE Compression Functions

File and Disk Handling Functions

Function - DiskFree

Declare: Public Declare Function **DiskFree** Lib "mslib145.dll" (ByVal Drive As String) As Double

Purpose: Returns the number of free bytes in the specified disk. The drive letter must be the first letter of the string. Case is not significant. You can specify only the drive letter if you wish. Uses MSDN API code.

Example:
```
Debug.Print DiskFree("C") ' Prints out 3309388  (bytes)
```

Functions - DiskFreeMeg and DiskFreeGig

Declares: Public Declare Function **DiskFreeMeg** Lib "mslib145.dll" (ByVal Drive As String) As Double
Public Declare Function **DiskFreeGig** Lib "mslib145.dll" (ByVal Drive As String) As Double

Purpose: As for DiskFree - these functions return the number of free Megabytes or Gigabytes in the specified disk. The drive letter must be the first letter of the string. Case is not significant. You can specify only the drive letter if you wish. Uses MSDN API code. 1 Megabyte here is 1024 x 1024 bytes. 1 Gigabyte is 1024 x megabytes. Note that hard-drive manufacturers may choose to specify a megabyte as 1,000,000 bytes.

Example:
```
Debug.Print DiskFreeGig("C")  ' Prints out 3.30  (bytes)
```

Function - DirExists

Declare: Public Declare Function **DirExists** Lib "mslib145.dll" (ByVal Filespec As String) As Boolean

Purpose: Checks to see if a directory exists. The directory attribute is specifically searched-for. Returns True if found, False if not

Notes: Directories are found regardless of hidden, system or readonly attributes
Any trailing slash is ignored internally. Note that the root directory of a drive always exists and is equivalent to **DriveExists**()
Note that paths which omit the root start such as "c:dirname" should be avoided since they will search for dirname in the current directory, not in c:\

Function - DriveExists

Declare: Public Declare Function **DriveExists** Lib "mslib145.dll" (ByVal Filespec As String) As Boolean

Purpose: Checks to see if a logical rather than a physical disk drive exists. The disk drive may exist as an entity but may not actually be a valid drive. i.e. it may be a virtual drive such as a removable MMC card-reader disk which no card inserted. Use **IsValidDisk**() to check if the disk is usable/operational.
Returns True if found, False if not

Notes: Checks to see if the drive is valid. Note that CDROMs and disconnected drives may return "invalid" although, technically they "exist" as physical or "disconnected" drives. Will not test for the presence of an empty CDROM or DVDROM drive

Function - FileCount

Declare: Public Declare Function **FileCount** Lib "mslib145.dll" (ByVal PathSpec As String) As Long

Purpose: Return a count of the number of matching files in a given directory

---

Function - FileExists

Declare: Public Declare Function **FileExists** Lib "mslib145.dll" (ByVal Filespec As String) As Boolean

Purpose: Tests to see if a file or filespec given by willdcards exists. Returns a Boolean (True or False) depending on whether the file or files are found

Examples:
```
Debug.Print FileExists("c:\boot.ini")
Debug.Print FileExists("c:\temp\*.tmp")
```

Comment: Wildcards are acceptable. A full pathspec is usually required
Directories are not handled by this function, only files

Use FileExists() to test for the presence of any disk drive. (DriveExists())
e.g. FileExists("Q:") returns False if Q: does not exist and True if it does

---

Function - FileLength

Declare: Public Declare Function **FileLength** Lib "mslib145.dll" (ByVal FileName As String) As Long

Purpose: Returns the size of the given file in bytes using the Windows API
See the Visual BASIC FileLen() function

Function - FileNameMatch

Declare: Public Declare Function **FileNameMatch** Lib "mslib145.dll" (ByVal FileName As String, ByVal Wildcard As String) As Boolean

Purpose: Match a filename by means of a simple wildcard-string.

The wildcards permitted are the same as those for MS-DOS or a CMD-console but slightly more flexible. The character "*" will stand for any group of characters. The wildcard character "?" will stand for any single matching character. Any other are matched on a case-insensitive basis.

Example:
```
FilenameMatch("my-project.txt","??-p*.txt")  True
FilenameMatch("winhelp32.exe","w*.exe")      True
FilenameMatch("winhelp32.exe","w*32.exe")    True
FilenameMatch("winamp.dll","*amp.d??")       True
FilenameMatch("nosuch.txt","*amp.d??")       False
```

Function - GetDiskSize

Declare: Public Declare Function **GetDiskSize** Lib "mslib145.dll" (ByVal DiskLetter As String, Optional ByVal CompensateBy10K As Boolean = True) As Currency

Purpose: Return the size of a disk drive in bytes.

Example: (For a hard-drive "C:" of size 30003503104 bytes)

```
Debug.Print GetDiskSize("C")   ' Colon is not mandatory
Returns:        30003503104    ' True size (integer)

Debug.Print GetDiskSize("C:",False)
Returns:        3000350.3104   ' True size/10,000 - Currency format
```

Notes: Note that this is calculated by the Windows API as an unsigned 64-bit for which there is no precise equivalent in VB5/6. In order to return an accurate value for values above 32-bits in size an **unsigned __int64** is returned which is interpreted via the declaration as a Currency data-type. Unfortunately in this case, there is an inbuilt bias in the Currency data-type which divides any value by 10,000. The declare automatically compensates for this by multiplying internally by 10,000. If, however, you are performing math where you have *already* compensated by this amount you can disable by specifying CompensateBy10K=False.

Function - GetDiskSizeGb

Declare: Public Declare Function **GetDiskSizeGb** Lib "mslib145.dll" (ByVal DiskLetter As String) As Double

Purpose: Return the size of a disk drive in gigabytes. (Disk size in bytes / (1024*1024*1024))
The value is returned as an decimal value in the form of a Double
You will usually need to use the Round() function to round this value.

Example: (For a hard-drive "C:" of size 30003503104 bytes)

```
Debug.Print GetDiskSizeGb("C")
Returns:        27.9429397583008

Debug.Print Round(GetDiskSizeGb("C"),3)
Returns:        27.943
```

See also: **CommaStr, GetDiskSize**

Function - GetDiskSizeMb

Declare: Public Declare Function **GetDiskSizeMb** Lib "mslib145.dll" (ByVal DiskLetter As String) As Long

Purpose: Return the size of a disk drive in megabytes. (Disk size in bytes / (1024*1024))
The value is returned as an integer value in the form of a Long

Example: (For a hard-drive "C:" of size 30003503104 bytes)

```
Debug.Print GetDiskSizeMb("C")
Returns:        28613
```

Notes: For more information see **GetDiskSize**()

Function - GetDiskType

Declare: Public Declare Function **GetDiskType** Lib "mslib145.dll" (ByVal DriveLetter As String) As Integer

Purpose: Provides a safe-alias for the Win32 API function **GetDriveTypeA**().
This function is used internally by **IsValidDrive**(), **IsCDROMDrive**() et. al.

The following values are returned as a valid or operational disk type:

0	DRIVE_UNKNOWN	Unknown drive type
1	DRIVE_NO_ROOT_DIR	The drive has no valid root directory
2	DRIVE_REMOVABLE	Removable type disk (USB/Floppy etc.)
3	DRIVE_FIXED	A hard-disk drive
4	DRIVE_REMOT	A mapped network drive
5	DRIVE_CDROM	Optical media such as DVD or CDROM
6	DRIVE_RAMDISK	RAM drive (e.g. ramdrive.sys)

Note that Constants for the above are not defined in the supplied module as you are expected to use the wrapper-functions - **IsHardDisk**(), **IsCDROMDisk**() etc.

Function - GetVolumeFileSystem

Declare: Public Declare Function **GetVolumeFileSystem** Lib "mslib145.dll" (_
ByVal RootDirString As String) As String

Purpose: Retrieves the file-system name for a given volume on a physical disk.
E.g. "FAT", "FAT32", "NTFS" etc. Note that the RootDirString parameter must indicate a root directory "\" for given drive. Full UNC paths are permitted.
The RootDirString parameter need only be an alpha character string e.g. "C"

Example:
```
Debug.Print GetVolumeFileSystem("C:\")
```

Result:
```
NTFS
```

Function - GetVolumeLabel

Declare: Public Declare Function **GetVolumeLabel** Lib "mslib145.dll" (_
ByVal RootDirString As String) As String

Purpose: Retrieves the file-system label for a given volume on a physical disk
Note that the RootDirString parameter must indicate a root directory "\" for given drive. Full UNC paths are permitted.
The RootDirString parameter need only be an alpha character string e.g. "C"

Example:
```
Debug.Print GetVolumeLabel("C:\")
```

Result:
```
Drive_C
```

Function - GetVolumeNameLength

Declare: Public Declare Function **GetVolumeNameLength** Lib "mslib145.dll" (_
ByVal RootDirString As String) As Long

Purpose: Retrieves the file-system maximum filename-length in bytes for a given volume on a physical disk. Note that the RootDirString parameter must indicate a root directory "\" for given drive. Full UNC paths are permitted.
The RootDirString parameter need only be an alpha character string e.g. "C"

Example:
```
Debug.Print GetVolumeNameLength("C:\")
```

Result:
```
255
```

Function - GetVolumeSerial

Declare: Public Declare Function **GetVolumeSerial** Lib "mslib145.dll" (_
ByVal RootDirString As String) As Long

Purpose: Retrieves the serial number as a VB Long for a given volume on a physical disk
Note that the RootDirString parameter must indicate a root directory "\" for given drive. Full UNC paths are permitted.
The RootDirString parameter need only be an alpha character string e.g. "C"

Example:
```
Debug.Print Hex$(GetVolumeSerial("C:\"))
```

Result:
```
3E220629
```

Function - IsCDROMDisk

Declare: Public Declare Function **IsCDROMDisk** Lib "mslib145.dll" (ByVal DriveLetter As String) As Boolean

Purpose: Test to see if a given drive letter is a valid CDROM, CDRW, DVDROM or other valid (active) optical drive.

Returns True if so otherwise False.

Function - IsHardDisk

Declare: Public Declare Function **IsHardDisk** Lib "mslib145.dll" (ByVal DriveLetter As String) As Boolean

Purpose: Test to see if a given drive letter is a valid (active) fixed hard-drive.

Returns True if so otherwise False.

Function - IsNetworkDisk

Declare: Public Declare Function **IsNetworkDisk** Lib "mslib145.dll" (ByVal DriveLetter As String) As Boolean

Purpose: Test to see if a given drive letter is a valid (active) mapped network drive.

Returns True if so otherwise False.

Function - IsRAMDisk

Declare: Public Declare Function **IsRAMDisk** Lib "mslib145.dll" (ByVal DriveLetter As String) As Boolean

Purpose: Test to see if a given drive letter is a valid (active) RAM drive such as that provided by RAMDRIVE.SYS or RAMDISK.SYS.

Returns True if so otherwise False.

Function - IsReady

Declare: Public Declare Function **IsReady** Lib "mslib145.dll" (ByVal DriveLetter As String) As Boolean

Purpose: Test to see if a given drive such as a USB or Floppy disk drive is ready to be read. This not only checks to see if the drive is valid and mapped to an active device but, where appropriate, whether or not media is inserted.

The DriveLetter parameter need only be an alpha character string e.g. "A"

Returns True if so otherwise False.

Function - IsRemovableDisk

Declare: Public Declare Function **IsRemovableDisk** Lib "mslib145.dll" (ByVal DriveLetter As String) As Boolean

Purpose: Test to see if a given drive letter is a removable-type drive such as a USB or Floppy disk drive.

Returns True if so otherwise False.

Function - IsSafeMode

Declare: Public Declare Function **IsSafeMode** Lib "mslib145.dll" () As Boolean

Purpose: Returns a Boolean indicating whether Windows is running in "Safe" mode

Returns True if so otherwise False.

Function - IsValidDisk

Declare: Public Declare Function **IsValidDisk** Lib "mslib145.dll" (ByVal DriveLetter As String) As Boolean

Purpose: Test to see if a given drive letter is a valid (active) disk of any kind.
Similar in function to DriveExists() but differs in that the disk being checked must be valid, active and available for use rather than an inactive logical drive. Does not indicate if the drive is ready for reading.

Returns True if so otherwise False.

Function - ListFiles

Declare: Public Declare Function **ListFiles** Lib "mslib145.dll" (ByVal PathSpec As String, _
Optional ByVal FileSpecs As String, _
Optional ByVal ArrayBase As Integer = 0, _
Optional ByVal IgnoreCase As Boolean = False) As Variant

Purpose: Create a directory-listing for filenames for a single folder. The result is returned in a String array as a Variant.

The returned Variant should be checked with the **IsEmpty**() function to see if any results were obtained. If successful, and the Variant is not "empty" then **Lbound**() should be used to get the lower array index and **Ubound**() used to get the upper-array index bounds. **Ubound**() will also indicate the number of items returned (the file-count). Unless you specify a value for the optional "ArrayBase" parameter the array-base will be set to zero regardless of any current **Option Base** setting in your program.

You can specify either an inclusive pathspec (with optional, trailing wildcard filename-specifier) as follows:

```
V=ListFiles("x:\some-path\")
V=ListFiles("c:\program files\google")
V=ListFiles("x:\some-other-path\*.exe")
V=ListFiles("d:\my-pictures\*.jpg")
V=ListFiles("\temp\*.tmp")
```

You may, additionally, specify a list of further wildcards which will filter, restrict or narrow-down this search to a group of matching files using a comma-delimited string. This enables complex searches to be performed. For example if you specify *.* in the PathSpec you can limit files of type "*.TXT" and "*.LOG". as follows.

```
V=ListFiles("x:\some-path\" , "*.txt, *.log")
V=ListFiles("c:\temp\*.*" , "*.tmp, vb*.*, *.log, *.txt, perflib*")
```

The list of restrictive/qualifying filespecs must be comma or space-delimited.
If you omit "*.*" and the path exists then it will be added automatically.

The returned array of files contains a list of filenames-only and does not include any path information. Case is not significant for filenames or searches and is ignored on Windows operating systems.

Example:

```
Dim v as Variant
Dim i as Integer       ' You can specify "Long" also
v=ListFiles("c:\photos\","*.jpg, *.jpeg, *.png, *.gif, *.bmp")
If IsArray(v) then
        For i=LBound(v) to Ubound(v)
                Debug.Print "File "; i; " is ["; v(i); "]"
        Next
EndIf
```

Notes: Recursion is not supported internally by this function although it could be used to construct a recursive function with. Bear in mind that both recursion and directory listing are CPU and memory-intensive.

See also: GetArrayDimensions

Function - MKDirs

Declare: Public Declare Function **MKDirs** Lib "mslib145.dll" (ByVal Path As String) _
As Integer

Purpose: Creates a directory or series of subdirectories in one operation

Notes: The VB "MKDir" function can only create one directory-segment at a time. This function will create an entire chain of subdirectories in one operation

The return value indicates the number of directory segments actually created (not the number of segments present). One or more of the left-most directory segments may already exist.

The path must be specified from the directory root. Either by using a backslash or by also specifying a drive prefix.

Example:

```
' Only c:\temp already exists
MKDirs "c:\temp\one\two\three\four"
```

Result:

```
4
```

See also: **MKTempName**

Function - MKTempName

Declare: Public Declare Function **MKTempName** Lib "mslib145.dll" (_
Optional ByVal Path As String = "", _
Optional ByVal FileType As String = ".tmp", _
Optional ByVal Length As Integer = 8, _
Optional ByVal Style As Integer = RandomStringAll) As String

Purpose: Creates a temporary filename randomly according to the details given

Description: The given path must be valid and is checked for existence against the live file system Any drive letter specified in the path must be valid. The length parameter can be any value up to the system maximum minus the length of the path segment and postfix filetype (if any).

Notes: If the Path parameter is omitted or is empty then the current TEMP or TMP environment value will be used instead.
If the FileType parameter is omitted or is empty then ".TMP" will be used
If the path is invalid and does not exist and the TEMP or TMP value cannot be retrieved then an empty string is returned.

The Path and FileType parameters are optional for the Declare/ANSI version
The Path and FileType parameters are mandatory for the TLB/Unicode version

The "Style" parameter must be a parameter from RandomStr() as follows:
The filename is guaranteed to be unique and not currently exist.

```
Public Const RandomStringMixedCase = 0
Public Const RandomStringLower = 1
Public Const RandomStringUpper = 2
Public Const RandomStringNumeric = 3
Public Const RandomStringAll = 4
Public Const RandomStringHex = 6
```

Example:

```
' Use the default system TEMP variable (no path given)

Debug.Print MKTempName("", "tmp", 8, RandomStringAll) ' Unicode/TLB
Debug.Print MKTempName        ' ANSI/Declare
```

Result:

```
c:\Temp\3LWIQ1K5.tmp
```

Example:

```
' Use an explicit path

Debug.Print MKTempName("c:\apps\", "$$$", 12, RandomStringNumeric)
```

Result:

```
c:\apps\650241428568.$$$
```

Function - ReadFileToString

Declare: Public Declare Function **ReadFileToString** Lib "mslib145.dll" (_
ByVal Filename As String, Optional ByVal Bytes As Long = 0, _
Optional ByRef ErrCode As Integer = 0) As String

Purpose: Reads a file into a a single buffer String.

Description: Primitive checks are made on the buffer before the file is read in.

Notes: Originally named ReadFile it has been renamed to avoid a possible clash with the Windows API function of the same name.

Memory is allocated using the Windows API

You may parse the file into multiple lines or a String() array using StrSplit as follows:

StrSplit(ReadFileToString("C:\boot.ini"),vbCrlf)

The ErrCode parameter is optional and may be ignored

Returns:

Code	Description
0	No error (success)
1	No filename specified (empty)
2	File was not found
3	Can't open the specified file
4	Can't allocate required memory

Example:

```
PrintR(StrSplit(ReadFileToString("c:\boot.ini"),vbCrlf))

ByVal:Array     Variant->String(6)
(
        [0] => "[boot loader]"
        [1] => "timeout=3"
        [2] => "default=multi(0)disk(0)rdisk(0)partition(2)\WINDOWS"
        [3] => "[operating systems]"
        [4] => "multi(0)disk(0)rdisk(0)partition(2)\WINDOWS="Mic...
        [5] => ""
)
```

See also: **PrintR, StrSplit**

Function - Unlink

Declare: Public Declare Function **Unlink** Lib "mslib145.dll" (ByVal Filename As String) As Long

Purpose: Replacement for the VB Kill <filename> command

Description: Provides a more functional substitute for Kill <filename>. Where it is essential to have more complete control over whether or not the file was removed and if not then know the reasons why then calling the "C" library "unlink()" function is offered here.

Returns:

Return Value	Description	Win32 API Error code
2	File Not found	ERROR_FILE_NOT_FOUND
32	File locked	ERROR_SHARING_VIOLATION
123	Invalid filename	ERROR_INVALID_NAME

See also: **WipeFile**

Function - WipeFile

Declare: Public Declare Function **WipeFile** Lib "mslib145.dll" (ByVal Filename As String, Optional ExtraSecure As Boolean = True) As Integer

Purpose: Securely wipes (erases) a given file by overwriting the file with random data.

Returns: On success WipeFile returns 0. On failure a non-zero value indicating the problem is returned. This enables your program to take further action as necessary. The return values from WipeFile are as follows:

Return Value	**Indicates**
0	Success
1	Invalid filename argument
2	File not found
3	Can't open for Wiping (file may be locked)
4	Can't unlink() (delete) the file

Notes: By default only up to the first 100Kb of a file are "scrubbed". If the ExtraSecure option is used then the whole file is scrubbed. Care should be taken as this may take a long time on large files.

WipeFile is useful for clearing temporary files which have been used for encryption etc.

See also: **Unlink**

Internet and Network Related Functions

Function - GetCGIArgs

Declare: Public Declare Function **GetCGIArgs** Lib "mslib145.dll" (ByVal s As String, v As Variant) As Integer

Purpose: Parse and break-up a CGI query-string passed via the server environment variable "QUERY_STRING".

Description: The value read for QUERY_STRING is passed as the first argument and an empty Variant as the second argument. The function returns the number of arguments found via the body and the variant returns a string-array of split argument pairs in the form "X=Y". The function IsEmpty() should be always used to test the returned Variant before use.

Example:

```
Dim V as Variant
Dim S as String
Dim I as Integer
S=Environ("QUERY_STRING")
If S<>"" Then
        I=GetCGIArgs(S,V)
        Debug.Print "Returned "; i " arguments"
        If IsArray(V) Then
                For i=LBound(V) To Ubound(V)
                        Debug.Print "Value "; i; "=["; V(i); "]"
                Next
        Endif
Endif
```

For a QUERY_STRING value "A=1&B=2&C=3" the code above prints out:

```
Returned 3 arguments
Value 1 =[A=1]
Value 2 =[B=2]
Value 3 =[C=3]
```

Notes: You may need to use URLDecode() to decode any encoded characters within the string. The delimiter characters, "?" and "&" are reserved and should not be encoded. If these characters are encoded within the string you intend to process then this should be decoded before calling GetCGIArgs()

Function - IPToLong

Declare: Public Declare Function **IPToLong** Lib "mslib145.dll" (ByVal IP As String) As Double

Purpose: Performs a text conversion to the equivalent of an "C" unsigned long value.

Notes: The unsigned product cannot be represented in VB by a signed long so the return is held as a double. Bear in mind that doubles are returned in the range of a "C" unsigned long - 0x00 to 0xFFFFFFF. The function needs to handle addresses from 0.0.0.0 to 255.255.255.255 (FF.FF.FF.FF) (0..+4294967295)

Examples: `IPToLong("192.168.2.1")`

Result: `3232236033`

See also: **LongToIP**

Function - IPMatch

Declare: Public Declare Function **IPMatch** Lib "mslib145.dll" (ByVal Mask As String, ByVal IP As String) As Boolean

Purpose: Matches the given IP address string to an IP "mask" string. The mask string may contain either an exact IP address or wildcards in the form of * or ? where * stands for any octet value of 1 to 3 digits which *may* exist and ? any individual octet digit which *must* exist. Partial masks are permitted as are mixtures of * and ? with numeric values.

For example 192.*.*.* will match all addresses starting with 192. whereas 192.??.*.* will match only addresses starting with 192 followed by a 2nd octet with *precisely* two digits.

Examples:

```
IPMatch("192.*.*.*","192.2.8.1")      returns True
IPMatch("192.??.*.*","192.2.8.1")     returns False
IPMatch("192.??.*.*","192.23.8.1")    returns True
IPMatch("192.","192.2.8.1")           returns True
IPMatch("*.*.*.*","192.2.8.1")        returns True
IPMatch("*","192.2.8.1")              returns True
IPMatch("???.?.?.?","192.2.8.1")      returns True
IPMatch("???.??.?.?","192.2.8.1")     returns True
IPMatch("???.2.?.?","192.2.8.1")      returns True
IPMatch("*.2.?.?","192.2.8.1")        returns True
IPMatch("*.2.2.2","192.2.8.1")        returns False
```

Comment: Partial-matching or partial wildcards are effective reading from left to right

See also: **MatchCIDR**

Function - LongToIP

Declare: Public Declare Function **LongToIP** Lib "mslib145.dll" (ByVal IP As Double) As String

Purpose: Converts a four-byte unsigned long value passed in VB as a double - such as the result of IPToLong(). This will be in the range 0x00 to 0xFFFFFFFF (0..4294967295)

Example: `LongToIP(IPToLong("192.168.2.1")) - Returns "192.168.2.1"`

Comment: Bear in mind that doubles are required as the full IP-range will result in a negative signed long integer value when held in a VB signed long.
Partial IP address strings such as "192." are accepted (equates to "192.0.0.0")

See also: **IPToLong**

Function - MapNetworkDrive

Declare: Public Declare Function **MapNetworkDrive** Lib "mslib145.dll" (_
ByVal ServerPath As String, _
ByVal DriveLetter As String, _
ByVal UserName As String, _
ByVal Password As String, _
ByRef ErrorCode As Long, _
Optional ByVal Persistent As Boolean = False, _
Optional OfferPromptForLogin As Boolean = False) As Boolean

Purpose: Connect a specific drive-letter to a shared network drive on a Microsoft or fully-Microsoft-compatible file server. Bear in mind that the process can be fairly complex and issues such as locked-accounts, credential conflict or non-existent resources need to be handled by the programmer calling these routines. It is recommended that the return code passed by-reference as the variable "ErrorCode" is checked and return values are appropriately handled.

Example:

```
Dim ServerPath as String
Dim UserName as String
Dim Password as String
Dim ErrorCode as Long
ServerPath="\\Comet\drive-c\docs"     ' Server name is \\Comet
UserName="Admin"
Password="secret"
If(MapNetworkDrive(ServerPath,"Q:",UserName,Password,ErrorCode) Then
        Debug.Print "Connected successfully"
Else
        Debug.Print "Failed to connect - return code was "; ErrorCode
EndIf
```

Notes: The servername should include prefix of "\\" e.g. "\\servername"

The server path should include the shared resource and may also include a sub-folder of that resource. Persistent connections can be specified if you wish by setting "Persistent" to True. These will persist within the user's profile into the next login-session. If desired Windows can be requested to ask for the user-credentials by setting "OfferPromptForLogin" to True

Potential problems will arise due to many errors including, but not limited to, the following examples:

Code	Reason
86	Wrong password
1219	Credential conflicts (already connected as a different user)
1909	Account is intruder-locked due to a bad-password

See also: **MapNextFreeNetworkDrive, UnMapNetworkDrive**

Function - MapNextFreeNetworkDrive

Declare: Public Declare Function **MapNextFreeNetworkDrive** Lib "mslib145.dll"
(ByVal ServerPath As String, _
ByVal UserName As String, _
ByVal Password As String, _
ByRef ErrorCode As Long, _
Optional ByVal Persistent As Boolean = False, _
Optional ByVal OfferPromptForLogin As Boolean = False) As String

Purpose: Connect the next available local drive letter to a shared network folder on a Microsoft or fully-Microsoft-compatible file server. Bear in mind that the process can be fairly complex. Issues such as locked-accounts, credential conflict or non-existent resources need to be handled by the programmer calling these routines. It is recommended that the return code passed by-reference as the variable "ErrorCode" is checked and return values are appropriately handled.

Example:

```
Dim ServerPath as String
Dim UserName as String
Dim Password as String
Dim ErrorCode as Long
Dim D as String
ServerPath="\\Comet\drive-c\docs"     ' Server name is Comet
UserName="Admin"
Password="secret"
D=""
D=MapNextFreeNetworkDrive(ServerPath,UserName,Password,ErrorCode)
If D<>"" Then
        Debug.Print "Connected successfully to drive "; D
Else
        Debug.Print "Failed to connect - return code was "; ErrorCode
EndIf
```

Notes: The server name should be specified with a double-backslash prefix "\\" - e.g. "\\servername".

The server path should include the shared resource by name and may also include a sub-folder of that same resource. Shared resources which are hidden from being browsed by being named with a terminating dollar sign.

Persistent connections can be specified if you wish by setting "Persistent" to True. These will persist within the user's profile into the next login-session.
If desired Windows can be requested to ask for the user-credentials by setting "OfferPromptForLogin" to True

Potential problems will arise due to many errors including, but not limited to, the following examples:

Code	Reason
86	Wrong password
1219	Credential conflicts (already connected as a different user)
1909	Account is intruder-locked due to a bad-password

See also: **MapNetworkDrive, UnmapNetworkDrive**

Function - UnmapNetworkDrive

Declare: Public Declare Function **UnmapNetworkDrive** Lib "mslib145.dll"
(ByVal DriveLetter As String, _
Optional ByVal Persistent As Boolean = False, _
Optional ByVal ForceIt As Boolean = False) As Long

Purpose: Disconnect a mapped network drive connected by MapNetworkDrive or MapNextFreeNetworkDrive. Windows may refuse to disconnect the drive if there are currently open files or network-searches pending. This may be overridden by setting "ForceIt" to True. If Persistent is set to True any changes will be reflected in the user-profile.

See also: **MapNetworkDrive, MapNextFreeNetworkDrive**

Function - MatchCIDR

Declare: Public Declare Function **MatchCIDR** Lib "mslib145.dll" (ByVal Mask As String,
ByVal IP As String) As Boolean

Purpose: A form of wildcard IP matching against a given IP mask. Often used for routing and IP-access filtering. Matches the given IP address against a CIDR mask and returns whether the match is made or not as a boolean (true or false). The mask must specify a bit value from 1 to 32 after the slash character which gives the number of "mask" bits to match to the IP address (reading from left to right).

Examples: MatchCIDR("192.168.0.0/16","192.168.2.12") ' Match 16 CIDR bits
Returns True
MatchCIDR("192.168.0.0/24","192.168.2.12") ' Match 24 CIDR bits
Returns False

Comment: The "mask" string must be in the format <ip-address>/<bitmask-value> with a separating forward-slash. The bitmask value must range from 1 to 32.
The function returns False on all errors. The caller must provide extended error-trapping and detection.
CIDR is an acronym for Classless Inter Domain Routing
(See http://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing)

See also: ***IPMatch***

Function - URLEncode

Declare: Public Declare Function **URLEncode** Lib "mslib145.dll" (ByVal s As String) As String

Purpose: Encodes a URL by encoding non-permitted characters as %xx trigraphs
This transformation is required on URLs submitted to webservers

See also: **URLDecode**

Function - URLDecode

Declare: Public Declare Function **URLDecode** Lib "mslib145.dll" (ByVal s As String) As String

Purpose: Decodes a URL which has been encoded by converting non-permitted characters to %xx trigraphs. This transformation is required on URLs which have been submitted to webservers and which are, for example, being processed by CGI applications which receive the encoded URL

Notes: URLs and query strings which are retrieved from a server environment table will almost always need to be transformed back into plain ASCII/ANSI text before their contents can be properly handled by any back-end application.

Important: You should call **ArgVal** before calling **URLDecode** on any CGI query string.
The reason for this is that the ampersand character (&) is used to delimit the query string and is therefore used to "tokenise" or split the string into an array. When the ampersand is used in text the character is encoded as %26 by the browser or web server. You should call URLDecode only after the array has been split otherwise the character will be wrongly interpreted as a delimiter character.

See also: **ArgVal, URLEncode**

Console Functions

No guarantee is given that these functions will work reliably on older versions of Windows although the console features have been found to work well on Windows NT4 (SP6a), Windows 2000 (SP3+) and XP (SP2+). Development is focused on supporting the entire Windows NT family where possible.

Sub - ClearConsoleAttributes

Declare: Public Declare Sub **ClearConsoleAttributes** Lib "mslib145" () Lib "mslib145" ()

Purpose: Resets console attributes set by SetConsoleAttributes back to the default.
The default value is grey text on a black background

See also: **SetConsoleAttributes**

Sub - CloseConsole

Declare: Public Declare Sub **CloseConsole** Lib "mslib145" ()

Purpose: Closes a console window opened by OpenConsole. No value is returned
Any pending input is lost. You should always call CloseConsole to close any console you have opened.

See also: **OpenConsole**

Sub - Cls

Declare: Public Declare Sub **Cls** Lib "mslib145" ()

Purpose: Clears the contents of a console window opened by OpenConsole.
No value is returned

Function - ConsoleOpen

Declare: Public Declare Function **ConsoleOpen** Lib "mslib145" () As Integer

Purpose: Returns True if the console was opened successfully and the handle to stdout successfully retrieved. Returns False once the console is closed.

Comments May be called at any time, but retrieving the handle from OpenConsole() is a preferable means of detection

See also: **CloseConsole, OpenConsole**

Function - ConsoleTitle

Declare: Public Declare Function **ConsoleTitle** Lib "mslib145" (ByVal s As String) As Long

Purpose: Sets the title of a console window which has been opened by OpenConsole. If no console is open yet then no output or error is generated. A string value is returned (which may be null or empty on failure)

See also: **CloseConsole, OpenConsole**

Function - ExitProgram

Declare: Public Declare Sub **ExitProgram** Lib "mslib145.dll" (ExitCode As Long)

Purpose: Exits the program back to the console and has the effect of an End statement but allows an ErrorLevel value to be returned to the Console Command Processor (CCP)

Notes: Should be used only for console-mode applications

Function - FlushConsole

Declare: Public Declare Function **FlushConsole** Lib "mslib145" () As Long

Purpose: Forces buffered console output issued by WriteLn or Writes to be flushed out to the console window. May be necessary to prevent "out-of-order" sequencing of console output. Equivalent to the "C" console I/O fflush() function but implemented using the Win32 API.

See also: **CloseConsole, OpenConsole, WriteLn, Writes**

Function - GetConsoleHandle

Declare: Public Declare Function **GetConsoleHandle** Lib "mslib145.dll" () As Long

Purpose: Retrieves the handle of an open console window for use with other Win32 API functions. This function returns the API result of GetActiveWindow() called a the time the console window was created with OpenConsole()

See also: **ConsoleOpen, OpenConsole**

Function - GetConsoleTitle

Declare: Public Declare Function **GetConsoleTitle** Lib "mslib145" Alias _
"getConsoleTitle" () As String

Purpose: Retrieves the title of a VBToolbox console window

See also: **ConsoleOpen, OpenConsole**

Sub - GotoXY

Declare: Public Declare Sub **GotoXY** Lib "mslib145" (ByVal x As Integer, ByVal y As Integer)

Purpose: Locates the cursor of an open console window at a particular X/Y coordinate

Notes: The console coordinates are "base 1". i.e. they start in the top L/H corner at 1,1

See also: **WhereX, WhereY**

Function - InKey

Declare: Public Declare Function **InKey** Lib "mslib145" () As Integer

Purpose: Checks to see if a keypress event is ready and if so returns the ASCII key-code of the keyboard character. Useful for monitoring loops awaiting a keypress

Example:

```
Dim k as integer
k=0
while k=0      ' Loop until a key is pressed
       k=InKey()
       'Do stuff here
Wend
```

See also: **ReadLn**

Function - InNativeConsole

Declare: Public Declare Function **InNativeConsole** Lib "mslib145" () As Boolean

Purpose: Returns true if the program is running as a native (converted) console application Returns false if unconverted and "as compiled" from VB

Comments This can be used as a means of checking if the program has been converted using LINK/EDITBIN into a native console app.

See also: **OpenConsole**

Function - IsCursorVisible

Declare: Public Declare Function **IsCursorVisible** Lib "mslib145.dll" () As Boolean

Purpose: Returns true if the cursor of an open VBToolbox console session is visible

See also: **SetCursorVisible**

Function - OpenConsole

Declare: Public Declare Function **OpenConsole** Lib "mslib145" () As Long

Purpose: Opens a new console window and returns the handle to it if successful and 0 if not. Only one console window can be opened and all output will be sent to that window. Use WriteLn and ReadLn to read and write to this window.
You should always call CloseConsole to close any console you have opened.

Notes: The long-standing problem of VB5 is of not being able to write to the current console using handle StdOut. Since VB is a Windows rather than a console application, n order to get a known handle from it's own window it must create it's own console window instance. Opening a new console for writing works fine for Webserver-based CGI processing and is exactly what would happen if you ran a console executable (EXE) from Windows Explorer.

OpenConsole() returns a Windows handle to the stdout stream if successful
This value can be stored and used to send output to stdout (the console) using other methods if desired.

On failure, INVALID_HANDLE_VALUE (-1. 0xffffffff) is returned

When an VB End statement is encountered the console is closed to output from VBToolbox and handles are released although any physical console window being written to will remain open until a CloseConsole command is issued. It will not be possible to write to any open console window once the End command has been met until another OpenConsole command is issued.

Example:

```
OpenConsole    ' If the EXE has not been relinked then
               ' a new console window will  be spawned
WriteLn "Hello World"         ' Writes the string followed by a CRLF
End                           ' Issue a GOTO Restart from the immediate
Restart:                      ' pane to resume output
WriteLn "Still there?"        ' Has no effect (window remains open)
OpenConsole                   ' Console reopened for output
WriteLn "Back again"          ' Output is now possible again
```

See also: **CloseConsole, ConsoleOpen, InNativeConsole**

Sub - Pause

Declare: Public Declare Sub **Pause** Lib "mslib145"
(Optional ByVal s As String = "Press a key...")

Purpose: Prints a message then waits for keyboard input.

The default message is `"Press a key ..."`

See also: **InKey, ReadLn**

Function - ReadLn

Declare: Public Declare Function **ReadLn** Lib "mslib145" (ByVal l As Long) As String

Purpose: Reads up to "l"characters from a console window which has been opened by OpenConsole. If no console is open yet then no output or error is generated. A string value is returned (which may be null or empty on failure)

See also: **InKey, Pause**

Sub - SetConsoleAttributes

Declare: Public Declare Sub **SetConsoleAttributes** Lib "mslib145" (ByVal Foreground As Integer, ByVal Background As Integer)

Purpose: Sets the text-attributes for a currently open console window.
Attributes should be set before sending text to the console
Use the supplied "C"-style colour constants to specify which colour
See the defined colour constants at the end of this section

See also: **ClearConsoleAttrbutes**

Sub - SetCursorVisible

Declare: Public Declare Sub **SetCursorVisible** Lib "mslib145.dll" (ByVal State As Boolean)

Purpose: Toggles the state of cursor visibility in an open VBToolbox console session

See also: **IsCursorVisible**

Function - WhereX

Declare: Public Declare Function **WhereX** Lib "mslib145" () As Integer

Purpose: Returns the X (horizontal) location of the cursor in an open console window
The location can be set using GotoXY()

See also: **GotoXY**

Function - WhereY

Declare: Public Declare Function **WhereY** Lib "mslib145" () As Integer

Purpose: Returns the Y (vertical) location of the cursor in an open console window
The location can be set using GotoXY()

See also: **GotoXY**

Function - WriteLn

Declare: Public Declare Function **WriteLn** Lib "mslib145" (ByVal s As String) As Long

Purpose: (Write-Line) Writes a text string out to a console window which has been opened by OpenConsole. If no console is open yet then no output or error is generated. WriteLn always appends a carriage-return (CRLF pair) and does not accept formatting. Use Format$() to format the output string.
WriteLn can be used with no parameters to issue a newline
The number of characters successfully written is returned.

See also: **FlushConsole, OpenConsole, ReadLn, Writes**

Function - Writes

Declare: Public Declare Function **Writes** Lib "mslib145" (ByVal s As String) As Long

Purpose: (Write String) Writes a text string out to a console window which has been opened by OpenConsole. If no console is open yet then no output or error is generated. Writes does NOT append a carriage-return (CRLF pair)
The number of characters successfully written is returned.

See also: **FlushConsole, OpenConsole, ReadLn, WriteLn**

Console Colour Constants

The following "C"-style constants are defined for use with SetConsoleAttributes() in the ANSI version of the library. Note that the names of these differ in the TLB/Unicode version where preset constants are provided. The TLB/Unicode version also offers dual English and American spelling.

```
Public Const BACKGROUND_BLACK = 0
Public Const FOREGROUND_BLACK = 0
Public Const FOREGROUND_BLUE = 1
Public Const FOREGROUND_GREEN = 2
Public Const FOREGROUND_CYAN = 3
Public Const FOREGROUND_RED = 4
Public Const FOREGROUND_MAGENTA = 5
Public Const FOREGROUND_YELLOW = 6
Public Const FOREGROUND_GRAY = 7        ' American spelling
Public Const FOREGROUND_GREY = 7        ' English (correct) spelling
Public Const FOREGROUND_INTENSITY = 8   ' Intensify the text colour

Public Const FOREGROUND_LIGHTBLUE = 1 Or FOREGROUND_INTENSITY
Public Const FOREGROUND_LIGHTGREEN = 2 Or FOREGROUND_INTENSITY
Public Const FOREGROUND_LIGHTCYAN = 3 Or FOREGROUND_INTENSITY
Public Const FOREGROUND_LIGHTRED = 4 Or FOREGROUND_INTENSITY
Public Const FOREGROUND_LIGHTMAGENTA = 5 Or FOREGROUND_INTENSITY
Public Const FOREGROUND_LIGHTYELLOW = 6 Or FOREGROUND_INTENSITY
Public Const FOREGROUND_WHITE = FOREGROUND_GREY Or FOREGROUND_INTENSITY

Public Const BACKGROUND_BLUE = 16
Public Const BACKGROUND_GREEN = 32
Public Const BACKGROUND_CYAN = 48
Public Const BACKGROUND_RED = 64
Public Const BACKGROUND_MAGENTA = 80
Public Const BACKGROUND_YELLOW = 96 ' Mustard
Public Const BACKGROUND_GRAY = 112
Public Const BACKGROUND_GREY = 112
Public Const BACKGROUND_INTENSITY = 128
Public Const BACKGROUND_WHITE = BACKGROUND_GRAY Or BACKGROUND_INTENSITY
```

See also: **SetConsoleAttributes**

Native Console App Conversion (EXE Conversion)

There is a simple modification you can optionally make to your compiled VB executable (EXE) file which will turn it into a full and native Win32 console application. One which does not open a new console window and which will interact normally with a standard command prompt and, for example, redirect or pipe it's output into a file.

The console functions of VBToolbox work just fine with such converted programs and need no special programming or changes. Just compile and do the following to your exe file. If run from Windows Explorer your program will open it's own console window otherwise it will reuse the existing one automatically.

Download Microsoft's linker program, LINK.EXE v5.12.8078 and associated EDTIBIN.EXE v5.12.8078. The best place I have found is to download it with MASM32 a freeware Macro-Assembler from http://www.masm32.com/.

First, note and accept the licence conditions and then install MASM32 to a folder on your hard drive. (You may also need to temporarily disable DEP (Data Execution Prevention) for the install to complete successfully). The version of LINK/EDITBIN bundled with VB5 (v4.20) won't work and oddly, nor will some higher versions such as v7.10 (MSVC++ 2008 Express edition). On incompatible versions the command gives warning LNK4044: unrecognised option "EDIT"; ignored;

This method of conversion has been tested with VBToolbox console code on Windows XP, Windows 2000 and Windows NT 4.0 (SP6a)

- Once you have installed MASM32 ...
- Compile your VB program which links to VBToolbox's console features.
  Ensure you have "Compile to native code" enabled in your Project options under Project->Properties->Compile Tab->"Compile to Native Code"
- Open a CMD prompt and change directory to the one which holds LINK and EDITBIN v5.12.8078 (should be in C:\MASM32\BIN\)
- Run the following command to convert your compiled EXE -
  LINK.EXE /EDIT /SUBSYSTEM:CONSOLE <filename>

  For example: for the vbcgi project, compile then run -
  LINK.EXE /EDIT /SUBSYSTEM:CONSOLE vbcgi.exe

  Alternatively, you can call EDITBIN.EXE directly using
  EDITBIN.EXE /SUBSYSTEM:CONSOLE <filename>
- You will see the following displayed on successful conversion (no detailed confirmation message)

  Microsoft (R) COFF Binary File Editor Version 5.12.8078
  Copyright (C) Microsoft Corp 1992-1998. All rights reserved.
- That's it. You can now test-run your EXE from a CMD window

Windows API-Related Functions

Includes some general file-handling functions which are not part of the Win32 API set.

Function - AddEventSource

Declare: Public Declare Function **AddEventSource** Lib "mslib145.dll" (ByVal AppName As String, Optional ByVal CategoryCount As Integer = 0) As Boolean

Purpose: Registers your application in the registry as a source of system event-log messages

Example:
```
Debug.Print "AddEventSource="; AddEventSource("VBToolbox")
```

Notes: You **MUST NOT** UPX or otherwise re-compress this DLL if you intend to use the logging functions. Although supplied UPX-compressed the DLL has to have very specific sections of the resource-table excluded.

You can use any name for "AppName" as long as it remains constant throughout the life of your application. Remove or "de-register" your application as a source of messages using RemoveEventSource(). You can still log events even when not registered or after you have de-registered your application. Your event messages will be prefixed with an informational warning by Windows to the effect that the message template could not be found.

The category-count should currently be omitted set to 0 (default) but the normal value would be 7. This is a bit-mapped value which ORs together values for each category (0x01, 0x02, 0x04). Currently categories are not fully-supported.

See also: **EventSource, RemoveEventSource, LogEvent**

Function - AddTrailingSlash

Declare: Public Declare Function **AddTrailingSlash** Lib "mslib145.dll" (ByVal FileName As String) As String

Purpose: Intelligently adds a trailing slash character ("\") to the end of a filename/path value This helps when you need to concatenate a filename onto the end of a path value and you would otherwise risk ending up with multiple backslash characters without a lot of checking. e.g. "c:\dir\" + "\" + "file.typ" might end up as "c:\dir\\file.typ".
No dynamic tests are made to distinguish between files or folders. The calling routine should test to see if a slash is being appended to the end of whichever type. For example: **"c:\fred"** - is this a file or a folder? The function will assume it is a folder and will append a slash character.
Use **GetNormalisedPath**() to normalise, check and correct partial or ambiguous paths.

Examples:
```
"c:" returns  "c:"
"c:\" returns "c:\"
"c:\fred" returns "c:\fred\"
"c" returns "c\"
""  returns ""
"q:\\" returns "q:\\"  (incorrect values are not corrected)
```

See also: **FileType**

Function - CanRedo

Declare: Public Declare Function **CanRedo** Lib "mslib145.dll" (ByVal hWnd As Long) _
As Boolean

Purpose: Returns a Boolean indicating whether a Win32 API "Redo" action can be performed using a EM_REDO SendMessage API call

See also: **CanUndo**

Function - CanUndo

Declare: Public Declare Function **CanUndo** Lib "mslib145.dll" (ByVal hWnd As Long)
As Boolean

Purpose: Returns a Boolean indicating whether a Win32 API "Undo" action can be performed using a EM_UNDO SendMessage API call

See also: **CanRedo**

Function - CreateGUID

Declare: Public Declare Function **CreateGUID** Lib "mslib145.dll" (Optional
ByVal Formatted As Boolean = True) As String

Purpose: Creates a 128-bit/16-byte/32 or 36-character Globally-Unique-Identifier (GUID) string in either plain-hex format or standard formatted format. A GUID has a very low-probability of ever being repeated and should never be repeated on the same machine as it is based around unique hardware signatures such as the unique network-card MAC address.

Example:

```
Debug.Print CreateGUID()      ' Default (True) is Windows-formatted
Debug.Print CreateGUID(false) ' Unformatted
```

Prints out:

```
"72b0f5dc-8f7d-4817-AD00-722B1CED6E9B"       (36 character string)
"d9f2832a7c114112967746717310C296"           (32 character string)
```

Function - FileType

Declare: Public Declare Function **FileType** Lib "mslib145.dll" (ByVal FileName As String) As String

Purpose: Returns the filetype part of a filename or filename/path.
This does not need to be in DOS 8.3 format. The filetype can be any length up to the Windows maximum path value

See also: **AddTrailingSlash**

Function - GetAppFileName

Declare: Public Declare Function **GetAppFileName** Lib "mslib145.dll" (Optional ByVal Handle as Long=0&) As String

Purpose: Retrieves the fully-qualified path name and filename of the current EXE program or a specified process by it's handle. This mirrors the functionality of App.Path in Visual BASIC for those not using VB to call VBToolbox.

Notes: This enables the launched program's precise location on disk to be determined

Example:
```
Debug.Print "GetAppFileName=";  GetAppFileName()
```

Result:
```
GetAppFileName=c:\apps\myprogram\myprogram.exe"
```

See also: **GetDLLFileName**

Function - GetCurrentDir

Declare: Public Declare Function **GetCurrentDir** Lib "mslib145.dll" () As String

Purpose: Retrieves the current working-directory

See also: **SetCurrentDir**

Function - GetDLLFileName

Declare: Public Declare Function **GetDLLFileName** Lib "mslib145.dll" () As String

Purpose: Retrieves the fully-qualified path name and filename of the current loaded DLL which is linked to your application.

Example:
```
Debug.Print "GetDLLFileName=";  GetDLLFileName()
```

Result:
```
GetDLLFileName=c:\windows\mslib145.dll"
```

See also: **GetAppFileName**

Function - GetError

Declare: Public Declare Function **GetError** Lib "mslib145.dll" () As Long

Purpose: Wrapper for Win32 API GetLastError function. Included for TLB use

Notes: GetError (GetLastError) returns the Windows error code which may be useful to reveal more information about an error or failure in an API call.

You can find the Windows System Error Codes here:
http://msdn.microsoft.com/en-us/library/ms681382(VS.85).aspx

Function - GetNormalisedPath

Declare: Public Declare Function **GetNormalisedPath** Lib "mslib145.dll"
(ByVal Path As String) As String

Purpose: Returns a corrected or "canonicalised" path from a partial or relative path
For example - inputting "\" will return "c:\", and "." will return the full, current directory including drive-letter. Note that Win32 API functions don't normally terminate folder/path strings with a backslash character other than for "root".

Example:
```
"c:" returns the current directory ("c:\vc5\myprojects\this-dll")
"\" returns "c:\"
"\temp" returns "c:\temp"
```

Function - GetOpenFile

Declare: Public Declare Function **GetOpenFile** Lib "mslib145.dll" (ByVal hWnd As Any,
ByVal Filter As String, ByVal Title As String, ByVal InitDir As String,
ByVal Flags As Integer) As String

Purpose: Provide an Open File dialog equivalent to that provided by the Visual BASIC Common Dialog OCX control. This used pure Win32 API and does not require any control to be installed or distributed with the program.

Notes: The Window handle (hWnd) may be specified as NULL (use 0&). However, if you do this then the Open File dialog won't be "bound" to your application and will not be "application modal" either. If you close your app then it will leave the file dialog still open. For flags values see the Win32 API reference for "GetOpenFileName"
All other values may also be left empty or set to zero e.g.
GetOpenFile(0&, "","","",0) - in which case defaults will be used.

See also: **GetSaveFile**

Function - GetProfileDir

Declare: Public Declare Function **GetProfileDir** Lib "mslib145.dll" (ByVal FolderFlags As Integer) As String

Purpose: Retrieves one of many common Windows directories.
E.g.: **"C:\Documents and Settings\All Users\Documents\My Pictures"**

Notes: Compatible with Versions of Windows from NT4.0 upwards

This function needs to be passed a valid CISDL value (Constant Item Special ID List) to identify the which class of folder to retrieve from Windows. Folders available will vary according to which version of Windows is in use. You can get more information from:
http://msdn.microsoft.com/en-us/library/bb762494(VS.85).aspx

Here are CSIDL values and folders from 0 to 255 for Windows XP SP2

No.	Folder Name
0	"C:\Documents and Settings\Admin\Desktop"
2	"C:\Documents and Settings\Admin\Start Menu\Programs"
5	"C:\Documents and Settings\Admin\My Documents"
6	"C:\Documents and Settings\Admin\Favorites"
7	"C:\Documents and Settings\All Users\Start Menu\Programs\Startup"
8	"C:\Documents and Settings\Admin\Recent"
9	"C:\Documents and Settings\Admin\SendTo"
11	"C:\Documents and Settings\Admin\Start Menu"
13	"C:\Documents and Settings\Admin\My Documents\My Music"
14	"C:\Documents and Settings\Admin\My Documents\My Videos"
16	"C:\Documents and Settings\Admin\Desktop"
19	"C:\Documents and Settings\Admin\NetHood"
20	"C:\WINDOWS\Fonts"
21	"C:\Documents and Settings\Admin\Templates"
22	"C:\Documents and Settings\All Users\Start Menu"
23	"C:\Documents and Settings\All Users\Start Menu\Programs"
24	"C:\Documents and Settings\All Users\Start Menu\Programs\Startup"
25	"C:\Documents and Settings\All Users\Desktop"
26	"C:\Documents and Settings\Admin\Application Data"
27	"C:\Documents and Settings\Admin\PrintHood"
28	"C:\Documents and Settings\Admin\Local Settings\Application Data"
31	"C:\Documents and Settings\All Users\Favorites"
32	"C:\Documents and Settings\Admin\Local Settings\Temporary Internet Files"
33	"C:\Documents and Settings\Admin\Cookies"
34	"C:\Documents and Settings\Admin\Local Settings\History"
35	"C:\Documents and Settings\All Users\Application Data"
36	"C:\WINDOWS"

37	"C:\WINDOWS\system32"
38	"C:\Program Files"
39	"C:\Documents and Settings\Admin\My Documents\My Pictures"
40	"C:\Documents and Settings\Admin"
41	"C:\WINDOWS\system32"
43	"C:\Program Files\Common Files"
45	"C:\Documents and Settings\All Users\Templates"
46	"C:\Documents and Settings\All Users\Documents"
47	"C:\Documents and Settings\All Users\Start Menu\Programs\Administrative Tools"
48	"C:\Documents and Settings\Admin\Start Menu\Programs\Administrative Tools"
53	"C:\Documents and Settings\All Users\Documents\My Music"
54	"C:\Documents and Settings\All Users\Documents\My Pictures"
55	"C:\Documents and Settings\All Users\Documents\My Videos"
56	"C:\WINDOWS\resources"
59	"C:\Documents and Settings\Admin\Local Settings\Application Data\Microsoft\CD Burning"

Function - GetSaveFile

Declare: Public Declare Function **GetSaveFile** Lib "mslib145.dll" (ByVal hWnd As Any, ByVal Filter As String, ByVal Title As String, ByVal InitDir As String, ByVal DefaultName As String, ByVal flags As Integer) As String

Purpose: Provide a Save File dialog equivalent to that provided by the Visual BASIC Common Dialog OCX control. This used pure Win32 API and does not require any control to be installed or distributed with the program.

Notes: The Window handle (hWnd) may be specified as NULL (use 0&). However, if you do this then the Open File dialog won't be "bound" to your application and will not be "application modal" either. If you close your app then it will leave the file dialog still open. For flags values see the Win32 API reference for "GetSaveFileName" A default filename may be given which will be returned if the user cancels input. All other values may also be left empty or set to zero e.g. GetSaveFile(0&, "","","","",0) - in which case defaults will be used.

See also: **GetSaveFile**

Function - GetSystemDir

Declare: Public Declare Function **GetSystemDir** Lib "mslib145.dll" () As String

Purpose: Returns the current windows system directory (e.g. **c:\windows\system32**)

Function - GetUserDir

Declare: Public Declare Function GetUserDir Lib "mslib145.dll" () As String

Purpose: Returns the current windows user directory (e.g."C:\Documents and Settings\Administrator")

Notes: Not compatible with Windows NT. Minimum Windows version: Windows 2000

Function - GetWallpaper

Declare: Public Declare Function **GetWallpaper** Lib "mslib145.dll" (Optional ByRef Style As Integer = 0) As String

Purpose: Retrieves the filename/path and, optionally, the display mode of the current wallpaper. The style value will be one of three values (if the parameter is used): 1:Centred (default), 2 (or 8):Tiled, 4:Stretched
The style value returned from GetWallpaper(), if specified, is designed to be used with SetWallpaper()

Example: GetWallpaper() ' Returns "c:\windows\winnt.bmp"
GetWallpaper(x) ' Also returns 8 in parameter "x" (tiled with both reg values set)

Notes: A JPG (JPEG) file cannot be used as a wallpaper. All such files need to be converted into a Windows Bitmap (BMP) file. This happens even on XP and Vista before they are used as a wallpaper. Converted JPEG wallpapers are usually stored in "\Documents and Settings\<User>\Local Settings\Application Data\Microsoft\"

See Also: **GetWallpaperStyle**

Function - GetWallpaperStyle

Declare: Public Declare Function **GetWallpaperStyle** Lib "mslib145.dll" () As Integer

Purpose: Return the current Windows wallpaper style setting. This will be one of "Centred" (Default), Tiled or Stretched - encoded as 1, 2 or 4 respectively.

Since two values are used in the registry to store the "Tiled" setting 8 is also returned for tiled mode. If both registry settings are found to be set then the return is encoded as (2 AND 8) - 10

See also: **GetWallpaper**

Function - GetWindowsDir

Declare: Public Declare Function **GetWindowsDir** Lib "mslib145.dll" () As String

Purpose: Returns the current windows directory (e.g."C**:\Windows\"**)

Function - IsClipboardEmpty

Declare: Public Declare Function **IsClipboardEmpty** Lib "mslib145.dll" () As Boolean

Purpose: Detects whether the Windows clipboard is empty or not

---

Function - IsEventSource

Declare: Public Declare Function **IsEventSource** Lib "mslib145.dll" (ByVal AppName As String) As Boolean

Purpose: Detects whether a given application (or application-string) is set in the system-registry as a known event-source by Windows. This function may be used as a conditional-test before calling **AddEventSource**() to register the application or to take other action to handle logging.

Example:
```
If Not IsEventSource("VBToolbox") Then
       Debug.Print "VBToolbox isn't set as an event-source, adding..."
       Debug.Print "AddEventSource=";AddEventSource("VBToolbox")
End If
```

Notes: Currently this function tests for the presence of the application-key in:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application\<App-name>
```

If found, then the key value for **EventMessageFile** is checked. If this is also present, then it's value is retrieved and the filename and full-path is compared with that of the loaded DLL. If the two match exactly (ignoring case) then the application is assumed to be registered with the system. Note that you can still log error messages using LogEvent() even if the application is not registered. See **LogEvent**() for more information.

See also: **AddEventSource, RemoveEventSource, LogEvent**

Function - IsMousePresent

Declare: Public Declare Function **IsMousePresent** Lib "mslib145.dll" () As Boolean

Purpose: Returns a Boolean indicating whether the system has a mouse connected

---

Function - IsNetworked

Declare: Public Declare Function **IsNetworked** Lib "mslib145.dll" () As Boolean

Purpose: Returns a Boolean indicating whether the system has an active network

---

Function - IsSafeMode

Declare: Public Declare Function **IsSafeMode** Lib "mslib145.dll" () As Boolean

Purpose: Returns a Boolean indicating whether Windows is running in Safe Mode or not

---

Function – IsSlowMachine

Declare: Public Declare Function **IsSlowMachine** Lib "mslib145.dll" () As Boolean

Purpose: Returns a Boolean indicating whether the CPU is a slow one

---

Function - LogEvent

Declare: Public Declare Function **LogEvent** Lib "mslib145.dll" (ByVal AppName As String, _
ByVal Message As String, _
ByVal LogType As Integer, _
Optional ByVal PostScript as String=vbNullString) As Boolean

Purpose: A convenient and direct API wrapper for Windows ("NT") family event logging into the system's Application event or error log. You may specify a main insertion string which will be placed inside the relevant message template. Additionally, you may specify a "postscript" string which will be concatenated onto the main string. There is a system-limit of 32k characters per insertion string. Commonly the postscript parameter may be used to append an error-code in string format.

Caution: The application needs to be registered as an event-source for the message descriptions to be readable by the Event Viewer *after* the event was logged and possibly after the program has exited. Until you register the VBToolbox DLL in the system registry as a known-message source using **AddEventSource**() the entries in the error-log will be readable but will contain a harmless warning prefix as follows:

```
The description for Event ID ( 11 ) in Source
( VBToolbox ) cannot be found. The local computer may
not have the necessary registry information or message
DLL files to display messages from a remote computer.
You may be able to use the /AUXSOURCE= flag to retrieve
this description; see Help and Support for details. The
following information is part of the event:
```

This also applies where a DLL or EXE has been registered as an event-source but was deleted from disk or renamed at a later date. Logging will still function should you not wish to add your application to the system registry. You should take care, however, modifying the system registry. Also, your program will need to be running as a user with sufficient privileges to add a new sub-key to:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application\
```

WARNING: You **MUST NOT** UPX or otherwise re-compress this DLL if you intend to use the logging functions. This applies even after the program has exited. Although supplied UPX-compressed, the DLL has to have very specific sections of the resource-table excluded.

Example:

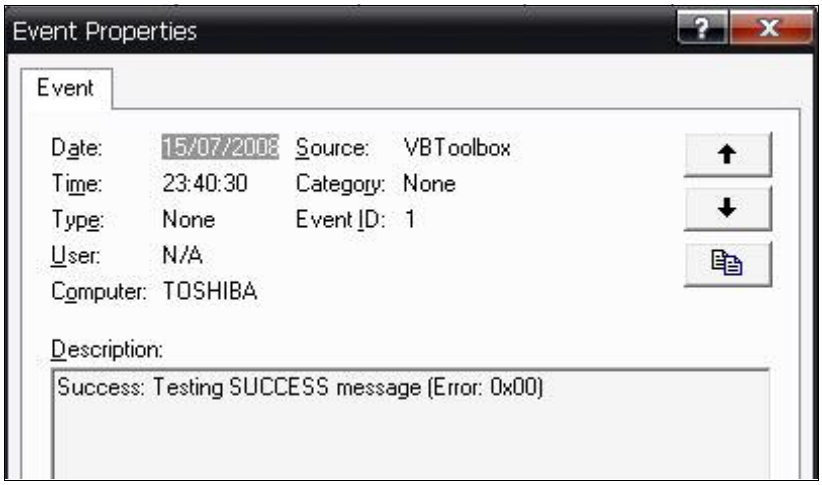
```
Debug.Print "AddEventSource="; AddEventSource("VBToolbox")
Debug.Print "LogEvent="; LogEvent("VBToolbox", _
       "Testing SUCCESS message", EVENTLOG_SUCCESS,"(Error: 0x00)")
```

Relevant registry-entries for "VBToolbox" are created under:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application\VBToolbox
```

Function - LogEvent (Continued...)

The log result is written to the Application log. (Start->Run->Eventvwr to view)



Notes: VB5 (and later) already offers the LogEvent method as an attribute of the App. object. VB5 logs an event in the application's log target. On Windows NT and later platforms, the method writes to the Windows Event log. On Windows 9x platforms, the method writes to the file specified in the LogPath property.

Note that you can format your messages by embedding carriage-return and linefeed pairs using vbCrlf to create line-breaks.

The event IDs map to keys held in the registry under:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application\<app-name>
```

Visual BASIC Object.Logevent() (Built-in)

```
vbLogEventTypeError           1       Error.
vbLogEventTypeWarning         2       Warning.
vbLogEventTypeInformation     4       Information.
```

Win32 API/VBToolbox Logevent()

```
Public Const EVENTLOG_SUCCESS = 0
Public Const EVENTLOG_ERROR_TYPE = 1
Public Const EVENTLOG_WARNING_TYPE = 2
Public Const EVENTLOG_INFORMATION_TYPE = 4
Public Const EVENTLOG_AUDIT_SUCCESS = 8
Public Const EVENTLOG_AUDIT_FAILURE = 10
```

You should use only the above constants when calling LogEvent()
The logged event-id will be +1 higher than the EVENTLOG_* const used.

For more information visit:
http://msdn.microsoft.com/en-us/library/aa363651(VS.85).aspx

Event-logging functionality has been tested only on the NT-family of Windows such as Windows XP. It has been tested on and is known to work on NT 4.0 (SP6a)
It has not been tested on any version of Windows 9x

See also: **IsEventSource(), AddEventSource(), RemoveEventSource()**

Function - MonitorCount

Declare: Public Declare Function **MonitorCount** Lib "mslib145.dll" () As Integer

Purpose: Returns an Integer giving the number of video monitors attached to Windows

---

Function - PrintDebug

Declare: Public Declare Sub **PrintDebug** Lib "mslib145.dll" (s As String)

Purpose: Sends a string to the registered system debugger using the OutputDebugString API call

---

Sub - PrintScreen

Declare: Public Declare Sub **PrintScreen** Lib "mslib145.dll" ()

Purpose: Captures the current application window contents to the clipboard

---

Function - RemoveEventSource

Declare: Public Declare Function **RemoveEventSource** Lib "mslib145.dll" (ByVal AppName As String) As Boolean

Purpose: De-Registers your application as a source of system event-log messages

Example:
```
Debug.Print "RemoveEventSource="; RemoveEventSource("VBToolbox")
```

Notes: You **MUST NOT** UPX or otherwise re-compress this DLL if you intend to use the logging functions. Although supplied UPX compressed the DLL has to have very specific sections of the resource-table excluded.

You can use any name for "AppName" as long as it remains constant throughout the life of your application. Add or "register" your application as a source of messages using **AddEventSource**(). You can still log events even when not registered or after you have de-registered your application. Your event messages will be prefixed with an informational warning by Windows to the effect that the message template could not be found.

Note that if you remove an event-source at any time *after* an event is logged then you will not be able to view the log-message in full using the message template within Eventvwr. Also, if the registered DLL or EXE is removed at a later point then the contents of the event log will be likewise affected.

See also: **IsEventSource, AddEventSource, LogEvent**

Function - SetCurrentDir

Declare: Public Declare Function **GetCurrentDir** Lib "mslib145.dll" _
(ByRef Path As String) As Boolean

Purpose: Sets the current working-directory

See also: **GetCurrentDir**

Function - SetWallpaper

Declare: Public Declare Function **SetWallpaper** Lib "mslib145.dll" (ByVal FileName
As String, Optional ByVal Style As Integer = 0) As Boolean

Purpose: Sets the wallpaper to a local bitmap (BMP) file. The optional value for display Style can be set to one of three values:
1:Centred (default), 2 (or 8):Tiled, 4:Stretched
Any other value selects the default. (Centred, normal size or "unstretched")
The value returned by GetWallpaper() may be used unchanged to set correctly.
A filename may be omitted and specified as an empty string "" to change the display mode for the current wallpaper.

Example:
```
SetWallpaper("C:\winnt\winnt.bmp",2)  ' Set to "winnt.bmp", tiled
SetWallpaper("",4)                    ' Change current to stretch format
```

Notes: The display modes are coded in binary increments of 1,2,4 and 8 as two registry settings are affected by the "tiled" display choice.
Note that although Windows will let you select a JPG (JPEG) file, the Windows API appears only to let you select a bitmap. What happens in Windows is that the files are converted to BMP format before use.

See also: **GetWallpaper, GetWallpaperStyle**

Function - ShellRun

Declare: Public Declare Function **ShellRun** Lib "mslib145.dll" (ByVal Command As String)
As Boolean

Purpose: Execute a Windows "Shell" command. This can be used to launch an associated file via the Windows Explorer shell interface. You can supply either the name of a file which has a valid association, a program, or a valid internet URL. The Window is opened in normal mode. Currently there are no options to execute the associated application hidden or minimised. This is merely a convenient wrapper for the ShellExecute() API function. If successfully-launched ShellRun returns True.

Examples:
```
ShellRun("c:\txt\somefile.txt")
ShellRun("program.exe")
ShellRun("http://www.google.com")
ShellRun(""mailto:someone@someisp.com?subject=Test Email&body=Hello")
```

Function - ShowFileProperties

Declare: Public Declare Function **ShowFileProperties** Lib "mslib145.dll" (_
ByVal s As String, _
Optional ByVal hWnd As Long = 0) As Boolean

Purpose: Launches a Windows Explorer "Properties" screen for a given file

Notes: The file must exist and the procedure performs relevant checks that the file exists before launching.

A Boolean value is returned indicating success or failure to launch

---

Function - WindowsSubVersion

Declare: Public Declare Function **WindowsSubVersion** Lib "mslib145.dll" () As String

Purpose: Returns the current windows sub-version as an ASNI string value -
e.g. returns the string “Service Pack 2” for Windows XP

Function - WindowsVersion

Declare: Public Declare Function **WindowsVersion** Lib "mslib145.dll" () As String

Purpose: Returns the current windows version as a 4-character (byte) string value:
e.g. "5.1" for Windows XP

Function - WindowsVersionMajor

Declare: Public Declare Function **WindowsVersionMajor** Lib "mslib145.dll" () As Long

Purpose: Returns the current windows version major number as a Long value:
e.g. 5 for Windows XP (5.1)

Function - WindowsVersionMinor

Declare: Public Declare Function **WindowsVersionMinor** Lib "mslib145.dll" () As Long

Purpose: Returns the current windows version minor number as a Long value:
e.g. 1 for Windows XP (5.1)

Function - WinSleep

Declare: Public Declare Sub **WinSleep** Lib "mslib145.dll" (ByVal Msecs As Integer)

Purpose: Halts execution for a specified number of milliseconds without halting the Operating System (Windows). This safely exposes the Windows API "Sleep()" function with bounds checking for negative numbers

Notes: The parameter should be given in thousandths of a second (milliseconds)
Values below 1 millisecond are rejected
The highest value possible is that of "signed int" milliseconds (32767ms) or 32.767 seconds

---

Windows Registry-Related Functions

Great care should be exercised when writing to the Windows registry. Writing improper data or to particular areas of the registry will render your system inoperative. It is highly-recommended that any code which modifies the registry ensures that the registry is securely backed-up first.

Windows Registry Constants

Note that the registry key prefix represents the "root" of the registry and any sub keys are relative to this. Hence, no sub key-value should be prefixed with a backslash "\" character. or the function-call will fail.

Right:
```
ReadStringFromRegistry(HKEY_LOCAL_USER, "Control Panel\desktop", _
                       "wallpaper")
```
Wrong:
```
ReadStringFromRegistry(HKEY_LOCAL_USER, "\Control Panel\desktop", _
                       "wallpaper")
```

These are the constants for the top-level keys. Unless correctly defined then you may run into problems when using the registry and Windows API.

The following values are negatively-signed Long Integers

```
Public Const HKEY_CLASSES_ROOT = &H80000000
Public Const HKEY_LOCAL_USER = &H80000001
Public Const HKEY_LOCAL_MACHINE = &H80000002
Public Const HKEY_USERS = &H80000003
Public Const HKEY_CURRENT_CONFIG = &H80000005
```

Function - ReadDWORDFromRegistry

Declare: Public Declare Function **ReadDWORDFromRegistry** Lib "mslib145.dll" (ByVal hKey As Long, ByVal SubKey As String, ByVal Value As String, ByRef DWORDValue As Long) As Boolean

Purpose: Read a double-word DWORD (Long) data type from a registry-key.

Notes: The function returns a boolean indicating whether the read was successful.
If False is returned any DWORD value returned should be ignored.
The user must have access to the particular section of the registry

---

Function - ReadStringFromRegistry

Declare: Public Declare Function **ReadStringFromRegistry** Lib "mslib145.dll" (ByVal hKey As Long, ByVal SubKey As String, ByVal Value As String) As String

Purpose: Reads a string from the registry in any of the top-level hives.
hKey represents one of the top level keys (See above)

Example:
```
ReadStringFromRegistry(HKEY_LOCAL_USER, "Control Panel\desktop", _
                       "wallpaper")
```

Result: `"c:\winnt\winnt256.bmp"`

Notes: You must be logged in as a user with adequate access-rights in order to be able to read (or write to) certain areas of the registry

---

Function - WriteDWORDToRegistry

Declare: Public Declare Function **WriteDWORDToRegistry** Lib "mslib145.dll" (ByVal hKey As Long, ByVal SubKey As String, ByVal Value As String, ByVal DWORDValue As Long) As Boolean

Purpose: Write a double-word DWORD (Long) data type to a registry-key.

Notes: The function returns a boolean indicating whether the write was successful.
If the key does not exist it will be created
The user-account running the program must have sufficient access-rights

---

Function - WriteStringToRegistry

Declare: Public Declare Function **WriteStringToRegistry** Lib "mslib145.dll" (ByVal hKey As Long, ByVal SubKey As String, ByVal Value As String, ByVal Data as String) As Boolean

Purpose: Writes a string from the registry in any of the top-level hives.
hKey represents one of the top level keys (See above)

Example:
```
WriteStringToRegistry(HKEY_CURRENT_USER, "Control Panel\desktop","wallpaper","My.bmp")
```

Notes: You must be logged in as a user with adequate access-rights in order to be able to read (or write to) certain areas of the registry
If the attempt to write was successful True will be returned

Later versions will include additional registry functions

Windows Process Functions

Function - GetPID

Declare: Public Declare Function **GetPID** Lib "mslib145.dll" () As Long

Purpose: Returns the unique Process ID (PID) of the current program as a long integer

---

Function - GetProcessMemoryUsed

Declare: Public Declare Function **GetProcessMemoryUsed** Lib "mslib145.dll" () As Long

Purpose: Returns the number of bytes from a process by process-ID. Use GetPID() to retrieve the process ID of your own application.

Notes: On failure **-1** is returned - otherwise the number of *bytes* used.

Divide by 1024 to get Kb, Divide by (1024.0*1024.0) or by 1048576.0 to return the number of megabytes (as a Double)

Example:
```
Debug.Print GetProcessMemoryUsed(GetPID())/1024; " Kb used"
```

Result:
```
63601 Kb used"
```

Graphics Functions

It is envisaged that very few graphics functions will be added other than specialised functions. If for no other reason than the free availability of the excellent mod_gd/GD graphics library from http://www.boutell.com/gd/ which can be called from most programming languages (including PHP).

A range of graphics functions have been used as an adjunct to encryption and as a container for data including RLE-compressed data produced by this library.

Function - BGRSplit

Declare: Public Declare Sub **BGRSplit** Lib "mslib145" (ByVal BGRValue As Long, ByRef Red As Integer, ByRef Green As Integer, ByRef Blue As Integer)

Purpose: Splits a Visual BASIC BGR colour value into separate Red, Green and Blue components. Note that VB stores 24-bit RGB values in reverse byte order.

Function - BGRToRGB

Declare: Public Declare Function **BGRToRGB** Lib "mslib145" (ByVal BGRValue As _ Long) As Long

Purpose: Perform a rapid transform using bit-shifting between a Visual BASIC BGR (Blue,Green,Red) encoded Long and normal RGB format. The higher byte of a 4-byte Long value is masked-off and ignored.

Example: Visual BASIC stores the product of the Rgb() function in BGR order. You can demonstrate this as follows.

```
Debug.Print Hex(Rgb(1,2,3))
Prints out: "30201"' 0x03, 0x02, 0x01

Debug.Print Hex(BGRToRGB(Rgb(1,2,3)))
Prints out: "10203"    ' 0x01, 0x02, 0x03
```

See also: **StringToBMP, BMPToString, BGRToRGB**

Function - GetColourSelection

Declare: Public Declare Function **GetColourSelection** Lib "mslib145.dll" (_
ByVal hWnd As Long, Optional FullExpand As Boolean = True, _
Optional RGBCurrentColour As Long = 0) As Long

Purpose: Evoke and use the standard Windows API colour selection dialogue

The current colour selection may be passed as a Microsoft (VB) RGB (BGR) colour value in "RGBCurrentColour"

Note that Microsoft 24-bit colour values are in Blue-Green-Red order from high to low byte. (BGR). You may use BGRToRGB to reverse the byte-order

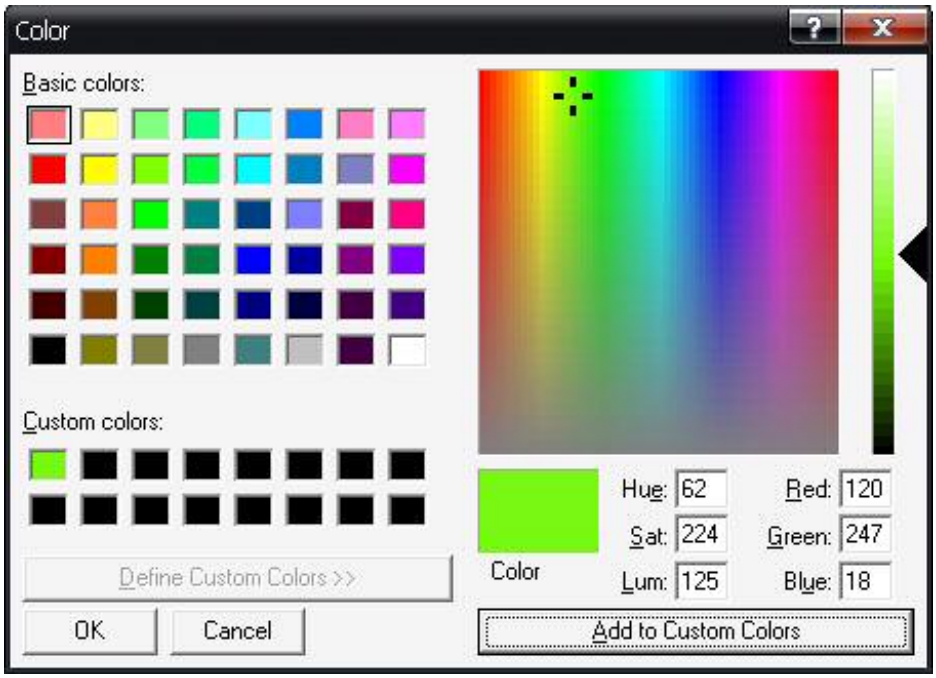
You may use the Visual BASIC Rgb() function to create a valid RGB Long value or input the value from say a Rich Text box control object.

By default the full colour-selection palette is expanded. You may contact this by setting the "FullExpand" parameter to False.

The 16 custom colour selections are persistent whilst the DLL is loaded in memory

Example: The following code gives an example of picking a colour using the dialogue

```
Debug.Print GetColourSelection(0,True,Rgb(255,128,8)):End
```



See also: **BGRToRGB, GetOpenFile, GetSaveFile**

Function - RGBVal

Declare: Public Declare Function **RGBVal** Lib "mslib145" (ByVal RGBString _
As String) As Long

Purpose: Convert a standard "internet" or HTML colour string value to an RGB Long data value.

Notes: The string need not be prefixed by the hash (#) character.

The data is in RGB (Red, Green, Blue) byte-order. Visual BASIC stores it's Rgb() function values in BGR (Blue, Green, Red) order

Example:
```
Debug.Print RGBVal("#0000FF") ' Blue
Prints out "255"

Debug.Print RGBVal("FF00FF")  ' Magenta
Prints out "16711935"
```

See also: **StringToBMP, BMPToString, BGRToRGB**

Function - StringToBMP

Declare: Public Declare Function **StringToBMP** Lib "mslib145" (ByVal s As String, _
ByVal Size As Long, _
ByRef BMPData As String, ByRef BMPSize As Long, _
Optional ByVal ForeColour As Long = 0, _
Optional ByVal BackColour As Long = &HFFFFFF) As Boolean

Purpose: Cryptography function. Converts any block of data, including valid bitmap data into a valid, rectangular 2-colour, bitmap (BMP) image. This may be useful as an efficient cryptograpic or other container for raw binary data. The bitmap header guarantees that the stored data will be returned precisely. and the 1 bit per pixel (8 pixels-per-byte) format guarantees efficient storage. The foreground and background colour may be set to the same value to visually "hide"any data stored in the image. The returned data-block may be written directly to file as a Windows BMP image.

The colour-values to be supplied are in VB BGR (Blue/Green/Red) byte-format as returned from the VB function, Rgb(). You can use BGRToRGB() to rotate between any byte-format.

The image will be rectangular and a multiple/minimum width of 32 pixels. This is in order to pack as much data efficiently as possible within the technical limitations of the BMP format which requires multiples of 4 bytes (32px at 1bpp)

The return value "BMPSize" is the size of the new bitmap file data-block in bytes. This is NOT the size of the encoded data held within the bitmap. It may include padding bytes added to make the width a multiple of 32 bits. The stored data block size can be obtained from BMPToString or BMPDataSize

This function does NOT encrypt or compress data. It simply offers a useful container object. Due to limitations of the bitmap format the image may not be perfectly square.

Notes: The size of the data-block should be supplied. This must NOT exceed the bounds of the allocated data-block. The returned length of the new bitmap object is returned via the optional parameter, BMPSize. Use BMPToString() to decode.

See also: **BMPInfo, BMPToString, BGRToRGB**

Function - BMPToString

Declare: Public Declare Function **BMPToString** Lib "mslib145" (_
ByVal BMPData As String, _
ByRef s As String, _
Optional ByRef NewStringLength As Long = 0) As Boolean

Purpose: Cryptography function. Decodes information stored in a bitmap (BMP) image created by StringToBMP. The size of the decoded data-block is returned via the optional parameter "NewStringLength".

Notes: Returns a Boolean value indicating success or failure

See also: **BMPInfo, StringToBMP, BGRToRGB**

Function - BMPDataSize

Declare: Public Declare Function **BMPDataSize** Lib "mslib145" (ByVal
BMPData As String) As Long

Purpose: Cryptography function. Returns the size of the data block (if any) stored within a BMP image by StringToBMP()

See also: **BMPInfo, StringToBMP, BGRToRGB**

Function - BMPInfo

Declare: Public Declare Function **BMPInfo** Lib "mslib145" (ByVal BMPData As String, _
Optional ByRef Width As Long, Optional ByRef Height As Long, _
Optional ByRef SizeBytes As Long, Optional ByRef Bpp As Long) As Boolean

Purpose: Retrieve rudimentary information about a bitmap (BMP) file. Namely, the Width in pixels, Height in pixels, the allocated data-block size in bytes and the number of bits per pixel. For 2 colour bitmaps this will be 1 bit per pixel or 8 pixels per byte.

See also: **StringToBMP, BMPToString, BGRToRGB**

Function - JPEGCheck

Declare: Public Declare Function **JPEGCheck** Lib "mslib145.dll" (ByVal FileName As String)
As Boolean

Purpose: Performs very basic checks on a JPEG file's header(s) and return a simple true or false value indicating that the file appears to be OK.

The file is not loaded, nor is any in-depth checking made on the entire file other than to ensure that the reported Windows size matches the readable number of bytes. The file may still be corrupted and unreadable for many other reasons such as a corrupted or short data-block. The function makes use of **JPEGHeader**()

See also: **JPEGHeader**

Function - JPEGHeader

Declare: Public Declare Function **JPEGHeader** Lib "mslib145.dll" (ByVal FileName As String, Width As Integer, Height As Integer, FileLength As Long) As Boolean

Purpose: Reads the pre-header and certain known headers of a JPEG file and makes basic checks on the file's integrity before returning the indicated file length and width and height dimensions in pixels. This gives a rough indication when a file is corrupted.

Note that the Width, Height and FileLength values are passed ByRef and this declaration should NOT be changed. All values must be passed even if they are not used. Use **JPEGCheck**() to perform simple file-integrity checks instead.
If the file appears to be readable "true" is returned. This does not, however, indicate that the file is fully-readable or guarantee that it is not corrupted due to damaged data-block or shortfall in the total number of bytes in a data-block.

A "False" return does not guarantee the file is faulty. Due to the wide-variety of JPEG formats it may be in a custom or bespoke format which is fully-readable by your system.

Example:

```
Dim Width as Integer
Dim Height as Integer
Dim Length as Long
Debug.Print JPEGHeader("myfile.jpg",Width,Height,Length)
Debug.Print "Width="; Width; "px"
Debug.Print "Height="; Height; "px"
Debug.Print "File length="; Length; " bytes"
```

Result:

```
True
1024px
768px
54332 bytes
```

See also: **JPEGCheck**

MAPI and Email Functions

Function - MAPISend

Declare: Public Declare Function MapiSend Lib "mslib145.dll" (ByVal hWindParent As Long, _
Optional ByVal strAttachmentName As String = "", _
Optional ByVal strSubject As String = "", _
Optional ByVal strMessage As String = "") As Boolean

Purpose: Send an email using Windows MAPI functionality. All values are optional.
Where hWndParent is omitted use zero (Long) as 0&

Requirements: A MAPI-compliant email client and MAPI32.DLL installed

Notes: This is experimental from V1.21+. The function requires that MAPI is correctly-configured on your Windows system and that you have a valid MAPI email client such as Microsoft Outlook, Outlook Express or Incredimail installed.

Using the Windows shell "MAILTO:" functionality can also be used where attachments are not required. Attachment support is not specified in the relevant RFCs and is implemented unreliably by differing mail clients.

At present there are bugs in the MAPI implementation of MAPI32.DLL in both SeaMonkey and Mozilla Thunderbird which mean file-attachments may not work properly or at all.

Note also that swapping default MAPI clients on Windows involves swapping out MAPI32.DLL which is a DLL customised to individual software. Changing this may break functionality in other installed clients (e.g. Novell Groupwise (TM)). Swapping MAPI clients may not always work well, or at all.

See:
https://bugzilla.mozilla.org/show_bug.cgi?id=244222
http://en.wikipedia.org/wiki/Messaging_Application_Programming_Interface

---

Compression Functions

These may be used in conjunction with image-manipulation container-functions such as StringToBMP, BMPToString etc. Bitmap images make ideal space-efficient containers for compressed and encrypted data.

Function - RLEByteCount

Declare: Public Declare Function **RLEByteCount** Lib "mslib145.dll" (ByVal sInput As String) As Long

Purpose: Returns the size of the compressed data-block held in an RLE-compressed string which has been compressed using RLECompress. If the data is not compressed then the function returns zero.

Function - RLECompress

Declare: Public Declare Function **RLECompress** Lib "mslib145.dll" (ByVal sInput _ As String, ByVal length As Long, Optional ByRef NewLength As Long) _ As String

Purpose: Compress an ANSI or binary-string using run-length encoding
The NewLength parameter is optional but will return the size of the newly-compressed string. The string to compress may contain nulls.

Notes: The length of compressed strings is stored by Visual BASIC but you should carefully-control and record the length of compressed strings in your own programs. Like many compression algorithms, RLE may result in no compression gain at all and could increase the size of the compressed block depending on the quality of the data supplied.

If the data is not compressible then zero is returned via NewLength and the original string is returned. You should test the NewLength return before attempting to decompress a string which may not have been compressed.. If the string is compressed successfully then the new, shorter

---

Function - RLECompressed

Declare: Public Declare Function **RLECompressed** Lib "mslib145.dll" (_ ByVal sInput As String) As Boolean

Purpose: Returns VB Boolean True if a string is compressed by the VBToolbox RLE compression functions. Otherwise False is returned.

Example: Immediate Pane example

```
b$="":a$="Helllo Worllld":call RLECompress(a$,b$):? RLECompressed(b$)
' Returns True
```

Function - RLECompressible

Declare: Public Declare Function **RLECompressible** Lib "mslib145.dll" (ByVal sInput As String) As Boolean

Purpose: Returns VB Boolean True if a compression-gain is possible using RLE encoding,

otherwise False. This function should be used to test before compressing otherwise compression could *increase* the size of the data block

---

Function - RLEUncompress

Declare: Public Declare Function **RLEUncompress** Lib "mslib145.dll" (ByVal sInput _
As String, ByVal length As Long, Optional ByRef NewLength As Long) As String

Purpose: Uncompress a binary string compressed using **RLECompress**

Notes: The length of compressed strings is stored by Visual BASIC but you should carefully-control and record the length of compressed strings in your own programs. Like many compression algorithms, RLE may result in no compression gain at all and could increase the size of the compressed block depending on the quality of the data supplied.

The size of the uncompressed string is returned via NewLengthFunction - RLECompressByteCount

---

Function - RLECompressByteCount

Declare: Public Declare Function **RLECompressByteCount** Lib "mslib145.dll" _
(ByVal sInput As String, ByVal length As Long) As Long

Purpose: Calculate the number of bytes required to store a given string or series of bytes This function can be used to decide whether a given block of data is worth compressing with **RLECompress**. RLECompressible serves the same purpose.

Function - RLEUncompressByteCount

Declare: Public Declare Function RLEUncompressByteCount Lib "mslib145.dll"
(ByVal sInput As String, ByVal length As Long) As Long

Purpose: Calculate the number of bytes required to store a given RLE-compressed string or series of bytes. This function can be used to determine the size of a buffer required to hold the result of **RLEUncompress**.

Visual BASIC Wrapper Code

Function - DLLVersion

Code:

```
Public Function DLLVersion() As String
    'Version is an integer in the format "101"%->"1.01"$
    Dim VerNum As Integer
    Dim Temp As String
    On Error GoTo NoDll

    Temp = "0.00"

    VerNum = LibVersion()
    Temp = Format(VerNum)
    If Len(Temp) = 3 Then
        Temp = Left$(Temp, 1) & "." & Right$(Temp, 2)
    End If
    DLLVersion = Temp
    Exit Function

NoDll:
    DLLVersion = "0.00"
    Resume Next
End Function
```

Purpose: Safely return the installed DLL version encoded in string format with decimal place Example v1.23 as the string "1.23"

Function - IsDLLInstalled

Code:

```
Public Function IsDLLInstalled() As Boolean
    Dim r As Long
    IsDLLInstalled = True
    On Error GoTo NoDll
    r = LibVersion()
    On Error GoTo 0
    Exit Function

NoDll:
    IsDLLInstalled = False
    Resume Next
End Function
```

Purpose: Detect whether the DLL is installed and available. Returns True/False
No version checking is performed this simply conforms if any version of the DLL is on the system. LibVersion() can be called to check the version number.

Visual BASIC Wrapper Code

Function - Base$

Code:

```
Public Function Base$(Value As Variant, BaseVal As Variant)
    Dim b As Byte   'Remove unsigned part
    b = CByte(BaseVal)
    Select Case (VarType(Value))
    Case vbByte:
        Base$ = StripTerminator(BaseConv(CByte(Value), b))
        'Debug.Print "BaseConv(Byte)"
    Case vbInteger:
        Base$ = StripTerminator(BaseConv(CInt(Value), b))
        'Debug.Print "BaseConv(Integer)"
    Case vbLong:
        Base$ = StripTerminator(BaseConv(CLng(Value), b))
        'Debug.Print "BaseConv(Long)"
    Case Default:
        'Debug.Print "BaseConv(Unknown)"
        Base$ = ""
    End Select
End Function
```

Purpose: Automatic overloading to call the correct base-conversion routines in the DLL
StripTerminator is now redundant and has been left in as an example

---

Function - Bin$

Code:

```
Public Function Bin$(x As Variant)
    'Debug.Print "VarType(x)="; VarType(x)
    Select Case (VarType(x))
    Case vbByte:
        Bin$ = Bin8(x)
        Debug.Print "Bin(Byte)"
    Case vbInteger:
        Bin$ = Bin16(x)
        Debug.Print "Bin(Integer)"
    Case vbLong:
        Bin$ = Bin32(x)
        Debug.Print "Bin(Long)"
    Case Default:
        Debug.Print "Bin(Unknown)"
    Bin$ = ""
    End Select
End Function
```

Purpose: Automatically call the correct overload for Bin() in the DLL

Visual BASIC Wrapper Code

Function - VariantToArray

Code:

```
Public Sub VariantToArray(ByRef V As Variant, ByRef R() As String)
    'Lbound will be affected by the "Option Base" setting
    On Local Error Resume Next
    Dim i As Long
    If IsArray(V) Then
        ReDim R(UBound(V))
        For i = LBound(V) To UBound(V)
            R(i) = V(i)
        Next
    End If
End Sub
```

Purpose: Converts a VB or VBToolbox string-array Variant into a dynamic VB String array. The supplied string array is destroyed and reassigned to hold the contents of the variant. The variant is unaffected. The size of the new string array can be determined using LBound() and UBound() as normal.

Notes: The ability to return strings via the function body is available from VB6 and higher only. VB5 requires that a reference to a string array is passed via the function body instead. The lower bound of the created string array will be affected by the use of the "Option Base" setting. Care should be taken that the current setting matches the Variant structure. "Option Base 0" is the safest setting to use.

Function - VBStr

Code:

```
Public Function VBStr(ByRef s As String) As String
   'Strips one single "C" terminating NULL (0x00) char from a C String
   'Only strips a terminator if there is one present
   Dim i As Integer
   i = InStr(s, Chr$(0))
   If i > 0 Then
      s = Left$(s, i - 1)
   End If
   VBStr = s
End Function
```

Purpose: Strips terminating NULL characters from returned "C" strings
Required for most functions which return string values prior to v1.11

Not normally required for versions 1.11and upwards unless strings return "binary" data

Visual BASIC Wrapper Code

Function - StripTerminator

Code:

```
Public Function StripTerminator(s As String) As String
    'Convert a NULL-terminated C++ string to VB
    'Calls VBStr - not defined by VB
    'This is just a wrapper function
    StripTerminator = VBStr(s)
End Function
```

Purpose: Wrapper duplicate of VBStr

End of Main Documentation

Appendix I - VBToolbox Visual BASIC Declares List

This is the list of declares specified in MSLIB145.BAS which is a companion to the both the ANSI and Unicode/TLB version of the Toolbox DLL. The supplied disk-file may contain more recent updates, additional useful comments and/or useful extra code

See MSLIB145.BAS for latest version

Appendx II - Erratum and Known or Reported Bugs

Early versions of _readfile (Alias) used the "C" fopen() function. However, it was found that using this would cause other functions to make VB hang. This was a difficult bug to track down. The problem has been resolved by replacing all stream-based i/o with the Windows API equivalents.

Feature Requests

If you have a feature-request you wish to be considered please send details to:
http://vbtoolbox.kerys.co.uk

Bug Reports and Erratum

Please send bug reports or erratum on this documentation via the website at
http://vbtoolbox.kerys.co.uk

Please include as much information as possible. Due to huge volumes of spam an email address will only be given for detailed reports in a limited number of cases. Bugs are resolved as quickly as possible enabling a new version of VBToolbox to be released.

Appendx III - Useful References and Links

VBToolbox website
http://vbtoolbox.kerys.co.uk

VBToolbox alternate URL
http://software.kerys.co.uk/vbtoolbox

Visual BASIC Discussion Forum
http://www.vbforums.com/

Visual BASIC Express Edition (Free)
http://www.microsoft.com/express/vb/

Information about the Free Visual BASIC 5 Custom Control Edition
http://support.microsoft.com/kb/q165524/

Working with "C" DLLs in Visual BASIC
http://www.fredshack.com/docs/DLL2VB.html

Using C++ DLLs in Visual BASIC (Mike D Sutton)
http://edais.mvps.org/Tutorials/CDLL/index.html

Visual BASIC Secrets (Kevin Wilson)
http://www.thevbzone.com/secrets.htm

FreeBASIC
http://www.freebasic.net/

VB Tutor
http://www.vbtutor.net/

ISO 8601 Date/Time Representations
http://www.mcs.vuw.ac.nz/technical/software/SGML/doc/iso8601/ISO8601.html

Date/Time Calendar Functions
http://www.wilsonmar.com/datepgms.htm

Dates and Times for Visual BASIC
http://ros.thevbzone.com/data_types_main.html

MicroApache - A portable USB Apache Webserver Distribution for Windows
http://microapache.kerys.co.uk

UPX A Free File-Compression Utility
http://upx.sourceforget.net/

EditPad - A Free RTF Wordprocessor with encryption provided via VBToolbox
http://editpad.kerys.co.uk

Free Software Page - More free software
http://software.kerys.co.uk/

Win32 API Error Codes
http://help.netop.com/support/errorcodes/win32_error_codes.htm

Mersenne Twister PRNG Home Page
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

Appendx IV - 3rd-Party Copyright Information

Information and acknowledgements pertinent to the MT Random functions used by this software.

Mersenne Twister Random Number Generator

The Mersenne Twister(MT) is a pseudorandom number generating algorithm developed by Makoto Matsumoto and Takuji Nishimura. The MT RNG functions include code which is copyright to the aforementioned and released as free/open-source software. More information including possible more efficient alternatives is available here: http://en.wikipedia.org/wiki/Mersenne_twister

Home page: http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

Contents

This page is intentionally left blank