

VB Toolbox

A Visual BASIC Toolbox DLL

© M Shaw - 2003-2009

<http://vbtoolbox.amadis.sytes.net/>

Documentation and
Programmer's Guide

Version 1.24

11th September 2009

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 the 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

This VBToolbox has undergone periodic testing and revision. Since the released code may contain more recent updates, please ensure that you refer to the current "declares" defined in MSLIB145.BAS 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://amadis.sytes.net/vbtoolbox>

All previous versions of this DLL should be discarded

This document was produced using OpenOffice 3- <http://www.openoffice.org>



General 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 rather than just by those learning to program in BASIC. Although the language has broad coverage 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.

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 is supplied and the library is primarily intended for educational use by the home hobbyist. It is not intended for commercial use and may not be distributed with commercial software without permission from the author. Non-commercial distribution is royalty-free and is permitted as long as the licence conditions are honoured. It may not be resold or distributed at a profit.

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.

Declares (Include File - Mslib145.BAS)

VB5 declares 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 or your VB program will crash, hang 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. Be warned.

Caution - UPX Compression

The DLL is 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 Windows will reference the disk-file-copy rather than the live, decompressed "in-memory" copy of the DLL in order to retrieve the event-strings. Consequently it will not be able to uncompress them properly. This will 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.

Caution - Thread Safety

This library should not be considered thread-safe. Apparently many aspects of VB5 and 6 and some aspects of SAFEARRAY code are not entirely thread-safe either and the presence of 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.

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 or it just won't work!

Caution - Function Parameters – Use Function Return Values

You should ensure you use the function body return rather than relying on a function parameter unless the documentation for that function recommends otherwise.

Unless stated the return value, particularly the case of functions handling and returning strings such as StripL(), is made via the function body. Functions of this library generally cannot be used as commands 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
StripL x	' Not permitted by this library –The VB syntax-checker should block this
y=Trim(x)	' Permitted by VB
y=StripL(x)	' Mandatory for all VBToolbox functions

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 - 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.

Caution - String Parameters - Use ByRef in Declares 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. **ByRef** uses a pointer to the string whereas **ByVal** makes a copy of **that** pointer. 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. Which is usually NOT what you want.

Strings which return "full binary" characters (0..255) outside the normal printable-range may need to be trimmed using the supplied Visual BASIC **VBS**`tr()` routine.

Is this a 16 or 32-bit Library?

This is a 32-bit only library. There is absolutely no demand for a 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.

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 "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 common reason why many C and C++ programs crash catastrophically after losing track of strings and writing into 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 then the null character needs to be stripped off to avoid a longer string than expected being processed by VB. A function is provided for this conversion process which has to be done in Visual BASIC - `VBS``tr()` and its 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.

NOTE: *Most functions from V1.11 onward will no longer require the use of `StripNulls()` or `VBS``tr()` 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.

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 something exported from a DLL called "EncryptFile64" using the function name "Encrypt64" then simply change or add an "Alias" declaration as follows.

```
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 in the program, 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:

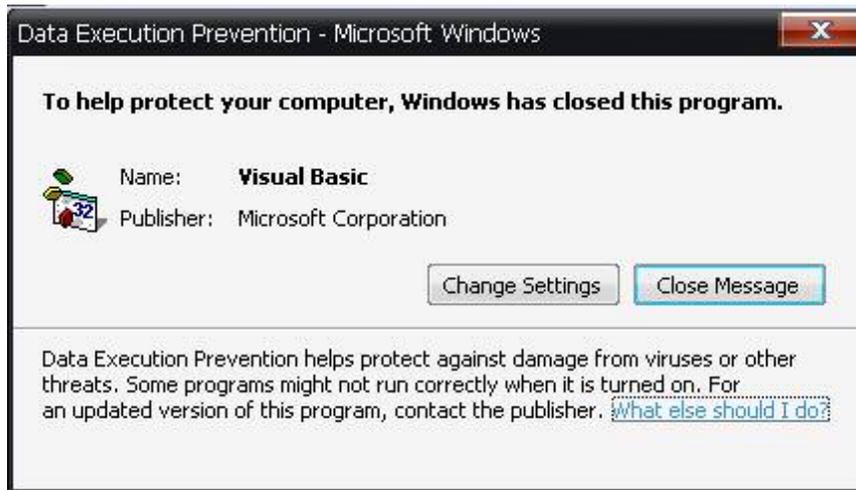
Notice that X varies randomly between each call...

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

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 for 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 create a new, standard 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 reqd.  
    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

VB5 does not 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. This is not a bug it is by design due to the limitations of the VB5/6 compiler.

Server CGI Applications

Despite not writing to any currently open CMD.EXE console window the console functionality works fine with Server CGI scripting allowing you to develop back-end CGI applications using Visual BASIC 5 or 6. 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 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

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 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 has been tested and appears to work correctly with the following versions of Windows

- Windows NT Workstation 4.0 (SP6a)
- Windows 2000 (SP4)
- Windows XP (SP2)
- Windows Vista

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 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.amadis.sytes.net>
<http://software.amadis.sytes.net/vbtoolbox/>

VBToolbox Installation

Installation Procedure

- 1) 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
- You may rename the DLL if you wish. Avoid multiple copies of the DLL on your system
- Copy the VBTOOLBOX.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
- That's it!. The functions are now available.

Installation Troubleshooting

- Note that console I/O may only work in Windows XP and Vista – it is untested in Win9x which is now obsolete and has a declining user-base.
- If you have problems then check you have downloaded the latest copies of both the BAS module and DLL.
- Check that you have them *both* 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 is by no means definitive. 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 - 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)

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()          'Make this common to all my DLLs
    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
    Temp = "0.00"

    VerNum = LibVersion()
    Temp = Format(VerNum)
    Temp = Left$(Temp, 1) & "." & Right$(Temp, 2)
    DLLVersion = Temp
End Function
```

String Handling Functions

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 using Tokenise() or GetArgs()

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

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

Prints out: "Argument B Found?=True"

See Also: ArgVal(), Tokenise() GetArgs(), GetCGIArgs(),

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. 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

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(), 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"

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"

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 - 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 or blanking a string.

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

Comments: 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.

Command - 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
```

Prints out: one
 two
 three

See Also: Tokenise(), ArgFound(), ArgVal()

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)
```

Prints out:

The dimensions of U are 0
The dimensions of V are 2
The dimensions of W are 1

Comments: 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

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 - 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
```

Comments: 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

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")
Returns "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 can 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

Function – Tokenise

Declare: Public Declare Function **Tokenise** Lib "mslib145.dll" (ByVal s As String, ByVal Tokens As String, Optional ByVal Base As Integer = 0) As Variant

Purpose: To break up a linear string into a two-dimensional string-array by means of a specified list of character tokens. This is similar to operations performed using the "C" strtok() function. This is a common and often complex problem to implement.

Comments: This function has an equivalent in VB6 called "Split()" 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 tokens specified is found.

If no token-separators are found, the original string will be returned in an array containing only one element. The "base" 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.

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, " ")
If Not IsEmpty(v) Then
    For i=LBound(v) to Ubound(v)
        Debug.Print v(i)
    Next
End If
```

Prints Out:

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

See Also: GetArgs(), ArgFound(), ArgVal(), GetCGIArgs()

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.

Comments: Characters are matched in more or less the same way 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.

Arithmetic and Number Functions

This includes various useful math and random-number generation functions

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()

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()

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

Function - PiStr

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

Comments: To be migrated to a separate PI.DLL

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.

Comments: 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.

Comments: 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
```

Comments: 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:

```
For 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:

```
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 the string "19680408" (R/H-excess OK)
UKtoISODate("08/04/68") returns "00000000" (2-digit years are not allowed)
UKtoISODate("8/4/68") returns "00000000" (single digits are not allowed)

Comments: 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"

Comments: 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"

Comments: 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"

Comments: 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)

Prints out "1st 2nd 3rd"

Comments: 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.
Errors: For dates outside the acceptable Unix epoch the string "ERR" is returned
- Example:** `Debug.Print PHPDate(Now, "r")`

Prints out "Fri, 20 Jun 2008 01:48:53 +0000"
- Comments:** **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 say -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

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()
- Comments:** 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\l\ \/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"
w	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\l\l\y \i\s !" 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 - VBDate

Declares: Public Declare Function **VBDate** 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 VBDate(Now)`

Prints out the string "39619.8489467593" (Fri, 20th June 2008 19:22:30)

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`

Prints out 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)`

Prints out "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#)`

Prints out: 62DFB1997D58E340

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#))`

Prints out: either 39619.9250115741 or "6/20/2008 10:12:01 PM" depending on the declare used.

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
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 ...

int	System dependent - but usually two bytes -32,768 to 32,767
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))`
Should print out "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))`
Prints out "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.

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

Prints out `66666666666.666`

Data Encoding Functions

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 0xFFFFFFFF (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 0xFFFFFFFF (4.50359963 × 10¹⁵) 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: (From the Visual BASIC debug screen type...)

```
For i=2 to 16: PRINT i, Base$(32767,i) : Next
```

(Outputs...)

2	1111111111111111
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

Eg. use Bin32(CLng(Val("&h00ffffff"))) rather than Bin32(Val("&h00ffffff"))

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.

Comments: 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.
Eg. use Bin\$(CLng(Val("&h00ffffff"))) rather than Bin\$(Val("&h00ffffff"))

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

Comments: 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 - 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.
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 (nested) to re-invert to the original value

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

Prints out: "9C"

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.
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 (nested) to re-invert to the original value

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

Prints out: "47E2"

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
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 (nested) to re-invert to the original value

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

Prints out: "77F632A1"

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

Prints out: "1A236F77"

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 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 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")`

Prints out: "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.

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.

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).

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.

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 (844424930131967). Values of 0x3FFF 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 24-bits (0xFFFFFFFF)

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...

```
0xFFFF FFFF FFFF = 281474976710655 (24-bits)
0x1 FFFF FFFF FFFF = 562949953421311 (25-bits)
0x2 FFFF FFFF FFFF = 844424930131967 (26-bits)
0x3 FFFF FFFF FFFF = 1.12589990684262E+15 (Google = 1.12589991 × 10^15)
0x4 FFFF FFFF FFFF = 1.40737488355328E+15 (Google = 1.40737488 × 10^15)
0x9 FFFF FFFF FFFF = 2.81474976710656E+15 (Google = 2.81474977 × 10^15)
0xA FFFF FFFF FFFF = 3.09622474381722E+15 (Google = 3.09622474 × 10^15)
0xF FFFF FFFF FFFF = 4.5035996273705E+15 (WinCalc) = 4503599627370495)
0xFF FFFF FFFF FFFF = 7.20575940379279E+16 (Google 7.2057594 × 10^16)
0xFFF FFFF FFFF FFFF = 1.15292150460685E+18 (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)
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"

Comments: 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 i 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"

Comments: A double is used to pass from VB in order to ensure that the correct unsigned range is passed from VB 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 (zips) 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.

Comments: 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

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 bytes.

Comments: Only strings which have been compressed by **BitPack** should be passed to this **BitUnPack** expands the length of a string by 100% (+/- 1 byte for odd values) function otherwise the resulting string will contain unpredictable contents. 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"

Function - EncodeString64

Declare: Public Declare Function **EncodeString64** Lib "mslib145.dll" (ByRef 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.

Comment: 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.

Function - DecodeString64

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

Comments: 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 ByVal 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))
```

Function - EncryptString

Declare: Public Declare Function **EncryptString** Lib "mslib145.dll" (ByVal s As String, _
ByVal length As Long, ByVal Password As String, Optional ByVal _
ExtraSecure As Boolean = True, _
Optional ByVal Salt As String) 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 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
```

Prints out:

```
j"%-W □u{=ø
Hello world
```

Comments: 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.

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

Comments: 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

Comments: 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 - 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 wildcards 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 - 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:

<code>FilenameMatch("my-project.txt", "??-p*.txt")</code>	True
<code>FilenameMatch("winhelp32.exe", "w*.exe")</code>	True
<code>FilenameMatch("winhelp32.exe", "w*32.exe")</code>	True
<code>FilenameMatch("winamp.dll", "*amp.d??")</code>	True
<code>FilenameMatch("nosuch.txt", "*amp.d??")</code>	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")
Returns:      30003503104   ' True size (integer)

Debug.Print GetDiskSize("C", False)
Returns:      3000350.3104   ' True size/10,000 - Currency format
```

Comments: 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 an **unsigned __int64** is returned which is interpreted via the declaration as a Currency data-type. Unfortunately 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
```

Comments: For more information see **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
```

Comments: 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_REMOTE	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 - 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 - 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 - 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. 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) 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 return 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 a 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. For example if you specify *.* in the PathSpec you can limit files of type "*.TXT" and "*.LOG". as follows.

```
V=ListFiles("x:\some-path\  
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\  
If Not IsEmpty(v)  
    For i=LBound(v) to Ubound(v)  
        Debug.Print "File "; i; " is ["; v(i); "]"  
    Next  
EndIf
```

Comments: 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.

Function - ReadFile

Declare: Public Declare Function **ReadFile** Lib "mslib145.dll" Alias "_readfile" (ByVal FileName As String, ByVal Buffer As String) As String

Purpose: Reads a file into a buffer string. Primitive checks are made on the buffer before the file is read in. ReadFile is a reserved library keyword. Because of this the library is exported as **_readfile** (lower-case with prefixed underscore) to avoid conflicts. You will need to declare this DLL with an "alias".

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". 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 Not IsEmpty(V) Then
        For i=LBound(V) To Ubound(V)
            Debug.Print "Value "; i; "=["; V(i); "]"
        Next
    Endif
Endif
```

For 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]
```

Comments: 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. This cannot be represented in VB by a signed long so the return is held as a double.

Examples: IPToLong("192.168.2.1")

Returns 3232236033

Comment: Bear in mind that doubles are returned in the range of a "C" unsigned long - 0x00 to 0xFFFFFFFF. The function needs to handle addresses from 0.0.0.0 to 255.255.255.255 (FF.FF.FF.FF) (0..4294967295)

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

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")

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
```

Comments: 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 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

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
```

Comments: 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 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

Function - UnmapNetworkDrive

Declare: Public Declare Function **UnmapNetworkDrive** Lib "mslib145.dll"
(ByVal DriveLetter As String, _
Optional ByVal Persistent As Boolean = False, _
Optional ByVal Forcelt 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 "Forcelt" to True. If Persistent is set to True any changes will be reflected in the user-profile.

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)

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

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.

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 (SP4) and XP (SP2). All development is focused on supporting the Windows NT family.

Command - 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

Command - 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 OpenConsole()

Command - CIs

Declare: Public Declare Sub **CIs** 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

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)

Command - 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

Comments: The console coordinates are "base 1". i.e. they start in the top L/H corner at 1,1

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 some stuff here
Wend
```

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.

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, in 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) is returned

Command - 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 ..."

Function - ReadLn

Declare: Public Declare Function **ReadLn** Lib "mslib145" (ByVal I As Long) As String

Purpose: Reads up to "I" 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)

Command - 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

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()

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()

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.

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.

Console Colour Constants

The following "C"-style constants are defined for use with SetConsoleAttributes()

```
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
```

Native Console App 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 EDITBIN.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: unrecognized 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")`

Comments: 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)
```

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

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()`

Prints out: `""GetDLLFileName=c:\windows\mslib145.dll"`

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.

Comments: 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.

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.

Comments: 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.

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 - 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)

Comments: 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\`"

Function - GetWallpaperStyle

Declare: Public Declare Function **GetWallpaperStyle** Lib "mslibl45.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

Function - GetWindowsDir

Declare: Public Declare Function **GetWindowsDir** Lib "mslibl45.dll" () As String

Purpose: Returns the current windows directory (e.g. c:\windows\)

Function - IsEventSource

Declare: Public Declare Function **IsEventSource** Lib "mslibl45.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
```

Comments: 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 - 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\
```

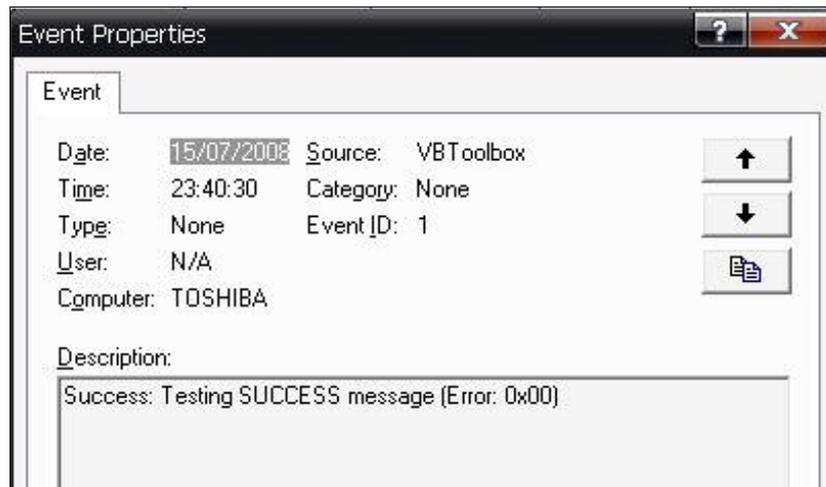
WARNING: You **MUST NOT** UPX or otherwise re-compress this DLL if you intend to use the logging functions. This applies even after the program has exited. Although supplied UPX-compressed, the DLL has to have very specific sections of the resource-table excluded.

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

The log result is written to the Application log. (Start->Run->Eventvwr to view)



Function - LogEvent (Continued...)

Comments: 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\

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](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 - 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")`

Comments: 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 - 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 - 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

Comments: 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.

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

Function - WindowsSubVersion

Declare: Public Declare Function **WindowsSubVersion** Lib "mslib145.dll" () As String

Purpose: Returns the current windows sub-version as an ANSI 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. 5.01 for Windows XP

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.

Comments: 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")`

Returns "c:\winnt\winnt256.bmp"

Comments: 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.

Comments: 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")`

Comments: 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.

Comments: 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"`

Prints out:

63601 Kb used"

Graphics Functions

It is envisaged that very few graphics functions will be added. 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).

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()**

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"
```

Prints out:

```
True
1024px
768px
54332 bytes
```

MAPI and Email Functions

Function - MAPISend

Declare: Public Declare Function MapiSend Lib "mslib145.dll" (ByVal hWndParent 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

Comments: 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.

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

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 - *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 NewLength

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**.

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$(x As Variant)
    Debug.Print "VarType(x)="; VarType(x)
    Select Case (VarType(x))
    Case vbByte:
        Bin$ = StripTerminator(Bin8(CByte(x)))
        Debug.Print "Bin(Byte)"
    Case vbInteger:
        Bin$ = StripTerminator(Bin16(CInt(x)))
        Debug.Print "Bin(Integer)"
    Case vbLong:
        Bin$ = StripTerminator(Bin32(CLng(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
StripTerminator is now redundant and has been left in as an example

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 VI.II
Not normally required for functions after VI.II 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 - Full VBToolbox Visual BASIC Declares List

This is the list of declares specified in MSLIB145.BAS. This module may contain more recent updates, additional useful comments and/or useful extra code

```
Option Base 0          ' Arrays are based at 0 (change if required)
Option Explicit       ' Vars must be declared

#Const VB5 = True    'Comment-out for VB6

Public Const VERNUM As String = "1.22" ' This test app's release version
Public Const DLL_VERSION As String = "1.22" ' This VB code release ID is required (to match DLL version)
Public Const WEBSITE As String = "http://vbtoolbox.amadis.sytes.net"
Public Const DLLNAME As String = "mslib145.dll" 'Can't use this Const in Declares unfortunately

Public Const HKEY_CLASSES_ROOT = &H80000000
'Public Const HKEY_LOCAL_USER = &H80000001 ' Alternate HKCU
Public Const HKEY_CURRENT_USER = &H80000001
Public Const HKEY_LOCAL_MACHINE = &H80000002
Public Const HKEY_USERS = &H80000003
Public Const HKEY_CURRENT_CONFIG = &H80000005

' RandomStr()
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 ' // v1.21+ // New
Public Const RandomStringBinaryString = 7 ' // V1.24+ - text string contining "0" and "1" characters
Public Const RandomStringBinaryStringBias0 = 8 ' // 75% bias towards producing "0" characters
Public Const RandomStringBinaryStringBias1 = 9 ' // 75% bias towards producing "1" characters

'For 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

'DLL management
Public Declare Function LibVersion Lib "mslib145.dll" () As Long

'Graphics functions
Public Declare Function JPEGHeader Lib "mslib145.dll" (ByVal FileName As String, _
    Width As Integer, Height As Integer, FileLength As Long) As Boolean
Public Declare Function JPEGCheck Lib "mslib145.dll" (ByVal FileName As String) As Boolean
Public Declare Function FileCount Lib "mslib145.dll" (ByVal PathSpec As String) As Long

'String functions
Public Declare Function WildcardMatch Lib "mslib145.dll" (ByVal s As String, _
    ByVal Wildcard As String, Optional IgnoreCase As Boolean = False) As Boolean
Public Declare Function Tokenise Lib "mslib145.dll" (ByVal s As String, _
    ByVal Tokens As String, Optional ByVal Base As Integer = 0) As Variant
Public Declare Function CommaStr Lib "mslib145.dll" (ByVal s As String, _
    Optional ByVal Decimals As Integer = 0) As String
Public Declare Function Comma Lib "mslib145.dll" (ByVal Value As Double, _
    Optional ByVal Decimals As Integer = 0) As String
Public Declare Function FillString Lib "mslib145.dll" (ByVal s As String, _
    ByVal CharToUse As String) As String
Public Declare Function InStr Lib "mslib145.dll" (ByVal s As String, ByVal search As String) As Long
Public Declare Function InChrRev Lib "mslib145.dll" (ByVal s As String, ByVal iChar As Integer) As Long
Public Declare Function InChr Lib "mslib145.dll" (ByVal s As String, ByVal iChar As Integer) As Long
Public Declare Function InStrRev Lib "mslib145.dll" (ByVal s As String, ByVal search As String) As Long
Public Declare Function IsAllChar Lib "mslib145.dll" (ByVal s As String, ByVal TheChar As String) As Boolean
Public Declare Function StripLStr Lib "mslib145.dll" (ByVal s As String, ByVal Mask As String) As String
Public Declare Function StripL Lib "mslib145.dll" (ByVal s As String, ByVal Mask As String) As String
Public Declare Function StripRStr Lib "mslib145.dll" (ByVal s As String, ByVal Mask As String) As String
Public Declare Function StripR Lib "mslib145.dll" (ByVal s As String, ByVal Mask As String) As String
Public Declare Function StrRev Lib "mslib145.dll" (ByVal s As String) As String
#If VB5 Then 'Replace() is built-in in VB6
Public Declare Function Replace Lib "mslib145.dll" (ByVal s As String, ByVal a As String, _
    ByVal b As String) As String
#End If
Public Declare Function ReplaceChar Lib "mslib145.dll" (ByVal s As String, _
    ByVal SearchedFor As String, ByVal Replacement As String) As String
Public Declare Function GetArrayDimensions Lib "mslib145.dll" (ByRef v As Variant) As Long

'Data conversion functions
Public Declare Function BaseConv Lib "mslib145.dll" (ByVal Value As Long, _
    ByVal Base As Integer) As String
Public Declare Function BaseConvDouble Lib "mslib145.dll" (ByVal Value As Double, _
    ByVal Base As Integer) As String
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
Public Declare Function BirToDec Lib "mslib145.dll" (ByVal s As String) As Long
Public Declare Function BitPack Lib "mslib145.dll" (ByVal s As String, _
    Optional ByVal Length As Long) As String
Public Declare Function BitUnPack Lib "mslib145.dll" (ByVal s As String, _
    Optional ByVal BitPairs As Long) As String
Public Declare Function RotateByte Lib "mslib145.dll" (ByRef b As Byte) As Byte
Public Declare Function RotateInt Lib "mslib145.dll" (ByRef i As Integer) As Integer
Public Declare Function RotateLong Lib "mslib145.dll" (ByRef l As Long) As Long
```

```

' Charhex, HexChar, HexInt, HexLong have changed names from V1.07+
Public Declare Function HexToChar Lib "mslib145.dll" (ByVal HexPair As String) As Byte
Public Declare Function HexToInt Lib "mslib145.dll" (ByVal Str As String) As Integer
Public Declare Function HexToLong Lib "mslib145.dll" (ByVal Str As String) As Double
Public Declare Function HexToDouble Lib "mslib145.dll" (ByVal Str As String) As Double
Public Declare Function CharToHex Lib "mslib145.dll" (ByVal i As Byte) As String
Public Declare Function IntToHex Lib "mslib145.dll" (ByVal i As Integer) As String
Public Declare Function LongToHex Lib "mslib145.dll" (ByVal i As Double) As String

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

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

Public Declare Function CreateGUID Lib "mslib145.dll" ( _
    Optional ByVal Formatted As Boolean = True) As String

'Legacy BASIC functions
'CV*/MK* Data Conversion - VB data types (byte sizes)
'Byte 1 byte 0 to 255
'Integer 2 bytes -32,768 to 32,767
'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

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
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
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
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

'Windows API functions
Public Declare Function logevent Lib "mslib145.dll" Alias "LogEvent" (ByVal AppName As _
    String, ByVal Message As String, ByVal LogType As _
    Integer, Optional ByVal PostScript As String = vbNullString) As Boolean
Public Declare Function IsEventSource Lib "mslib145.dll" (ByVal AppName As String) _
    As Boolean
Public Declare Function AddEventSource Lib "mslib145.dll" (ByVal AppName As String, _
    Optional ByVal CategoryCount As Integer = 0) As Boolean
Public Declare Function RemoveEventSource Lib "mslib145.dll" (ByVal AppName As String) _
    As Boolean
Public Declare Function DiskFree Lib "mslib145.dll" (ByVal Drive As String) As Double
Public Declare Function DiskFreeGig Lib "mslib145.dll" (ByVal Drive As String) As Double
Public Declare Function DiskFreeMeg Lib "mslib145.dll" (ByVal Drive As String) As Double
Public Declare Function FileExists Lib "mslib145.dll" (ByVal Filespec As String) As Boolean
Public Declare Function DriveExists Lib "mslib145.dll" (ByVal Filespec As String) As Boolean
Public Declare Function DirExists Lib "mslib145.dll" (ByVal Filespec As String) As Boolean
Public Declare Function GetAvailablePhysicalMemory Lib "mslib145.dll" () As Double
Public Declare Function GetTotalPhysicalMemory Lib "mslib145.dll" () As Double
Public Declare Function GetAvailableSwapfileMemory Lib "mslib145.dll" () As Double
Public Declare Function GetTotalSwapfileMemory Lib "mslib145.dll" () As Double
Public Declare Function GetAvailableVirtualMemory Lib "mslib145.dll" () As Double
Public Declare Function GetTotalVirtualMemory Lib "mslib145.dll" () As Double
Public Declare Function GetSystemDir Lib "mslib145.dll" () As String
Public Declare Function GetWindowsDir Lib "mslib145.dll" () As String
Public Declare Function WindowsSubVersion Lib "mslib145.dll" () As String
Public Declare Function WindowsVersion Lib "mslib145.dll" () As String
Public Declare Function GetLocalDriveStrings Lib "mslib145.dll" () As String
Public Declare Function GetDLLFileName Lib "mslib145.dll" () As String
Public Declare Function ShellRun Lib "mslib145.dll" (ByVal Command As String) _
    As Boolean
Public Declare Function GetDiskSize Lib "mslib145.dll" (ByVal DiskLetter As String, _
    Optional ByVal CompensateBy10K As Boolean = True) As Currency
Public Declare Function GetDiskSizeMb Lib "mslib145.dll" (ByVal DiskLetter As String) _
    As Long
Public Declare Function GetDiskSizeGb Lib "mslib145.dll" (ByVal DiskLetter As String) _
    As Double

'Alias for Win32 Sleep() with sanity-checks for parameters
Public Declare Sub WinSleep Lib "mslib145.dll" (ByVal Msecs As Integer)
Public Declare Function SetWallpaper Lib "mslib145.dll" (ByVal FileName As String, _
    Optional ByVal Style As Integer = 1) As Boolean ' Default style is centred
Public Declare Function GetWallpaper Lib "mslib145.dll" () As String
Public Declare Function GetWallpaperStyle Lib "mslib145.dll" () As Integer

Public Declare Function ReadStringFromRegistry Lib "mslib145.dll" (ByVal hKey As Long, _
    ByVal SubKey As String, ByVal Value As String) As String
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
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
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

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
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

Public Declare Function GetDiskType Lib "mslib145.dll" (ByVal DriveLetter As String) As Integer
Public Declare Function IsCDROMDisk Lib "mslib145.dll" (ByVal DriveLetter As String) As Boolean
Public Declare Function IsHardDisk Lib "mslib145.dll" (ByVal DriveLetter As String) As Boolean

```

```

Public Declare Function IsNetworkDisk Lib "mslib145.dll" (ByVal DriveLetter As String) As Boolean
Public Declare Function IsRAMDisk Lib "mslib145.dll" (ByVal DriveLetter As String) As Boolean
Public Declare Function IsRemovableDisk Lib "mslib145.dll" (ByVal DriveLetter As String) As Boolean
Public Declare Function IsValidDisk Lib "mslib145.dll" (ByVal DriveLetter As String) As Boolean

' File functions
Public Declare Function AddTrailingSlash Lib "mslib145.dll" (ByVal FileName As String) As String
Public Declare Function FilenameMatch Lib "mslib145.dll" (ByVal FileName As String, _
    ByVal Wildcard As String) As Boolean
Public Declare Function FileType Lib "mslib145.dll" (ByVal FileName As String) As String
Public Declare Function GetNormalisedPath Lib "mslib145.dll" (ByVal Path As String) As String
Public Declare Function ListFiles Lib "mslib145.dll" (ByVal PathSpec As String, _
    Optional ByVal FileSpecs As String, Optional ByVal ArrayBase As Integer = 0) As Variant
Public Declare Function ReadFile Lib "mslib145.dll" Alias "_readfile" (ByVal FileName As String, _
    Optional ByVal Bytes As Long) As String

'Date and time functions
Public Declare Function ISOToUKDate Lib "mslib145.dll" (ByVal YYYYMMDD As String) As String
Public Declare Function UKToISODate Lib "mslib145.dll" (ByVal DDSMMsYYYY As String) As String
Public Declare Function UKShortToISODate Lib "mslib145.dll" (ByVal DDSMMsYY As String) As String
Public Declare Function IsLeapYear Lib "mslib145.dll" (ByVal Year As Integer) As Boolean
Public Declare Function NumOrd Lib "mslib145.dll" (ByVal N As Integer) As String
Public Declare Function PHPDate Lib "mslib145.dll" (ByVal d As Date, _
    Optional ByVal s As String) As String
Public Declare Function PHPDateNow Lib "mslib145.dll" (Optional ByVal s As String) As String
Public Declare Function VBDate Lib "mslib145.dll" (ByVal d As Date) As String
Public Declare Function VBDateMsecs Lib "mslib145.dll" (ByVal d As Date) As Integer
Public Declare Function VBDateToCTime Lib "mslib145.dll" (ByVal d As Date) As Double
Public Declare Function CTimeToVBDate Lib "mslib145.dll" (ByVal d As Double) As Date
Public Declare Function DSTAdjust Lib "mslib145.dll" (ByVal d As Date) As Double
Public Declare Function DateToHex Lib "mslib145.dll" (ByVal d As Date) As String
Public Declare Function HexToDate Lib "mslib145.dll" (ByVal s As String) As Double
'Alternate declaration
'Public Declare Function HexToDate Lib "mslib145.dll" (ByVal s As String) As Date

'Maths functions
Public Declare Function Gcd Lib "mslib145.dll" (ByVal a As Long, ByVal b As Long) As Long
Public Declare Function Max Lib "mslib145.dll" (ByVal a As Integer, _
    ByVal b As Integer) As Integer
Public Declare Function Min Lib "mslib145.dll" (ByVal a As Integer, _
    ByVal b As Integer) As Integer
Public Declare Function Integral Lib "mslib145.dll" (ByVal d As Double) As Double
Public Declare Function Fraction Lib "mslib145.dll" (ByVal d As Double) As Double
#If VB5 Then 'VB5 et. al.
Public Declare Function Round Lib "mslib145.dll" (ByVal d As Double, _
    Optional ByVal Places As Integer) As Double
#Else
'Alias for VB6 etc.
Public Declare Function VBRound Lib "mslib145.dll" Alias "Round" (ByVal d As Double, _
    Optional ByVal Places As Integer) As Double
#End If

'IP address Network and Internet
Public Declare Function IPMatch Lib "mslib145.dll" (ByVal Mask As String, _
    ByVal IP As String) As Boolean
Public Declare Function IPToLong Lib "mslib145.dll" (ByVal IP As String) As Double
Public Declare Function LongToIP Lib "mslib145.dll" (ByVal IP As Double) As String
Public Declare Function MatchCIDR Lib "mslib145.dll" (ByVal Mask As String, _
    ByVal IP As String) As Boolean
Public Declare Function URLEncode Lib "mslib145.dll" (ByVal s As String) As String
Public Declare Function URLDecode Lib "mslib145.dll" (ByVal s As String) As String
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
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
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
Public Declare Function DisconnectFromServer Lib "mslib145.dll" (ByVal ServerName As String, _
    Optional ByVal ForceIt As Boolean = False) As Long
Public Declare Function GetCGIArgs Lib "mslib145.dll" (ByVal s As String, _
    ByRef v As Variant) As Integer
Public Declare Function ArgFound Lib "mslib145.dll" (ByRef v As Variant, ByVal s As String, _
    Optional ByVal IgnoreCase As Boolean = True) As Boolean
Public Declare Function ArgVal Lib "mslib145.dll" (ByRef v As Variant, ByVal s As String, _
    Optional ByVal IgnoreCase As Boolean = True) As String

'Encoding
Public Declare Function Base64Filter Lib "mslib145.dll" (ByVal s As String) As String
'Note: Params below have changed from v1.02 - strings should now be ByRef for BSTR DLL calls
Public Declare Function EncodeString64 Lib "mslib145.dll" (ByRef sInput As String, _
    Optional ByVal WrapWidth As Integer = 0) As String
'Public Declare Function DecodeString64 Lib "mslib145.dll" (ByRef sInput As String) As String
Public Declare Function DecodeString64 Lib "mslib145.dll" (ByRef sInput As String, _
    ByRef NewLength As Long) As String

'Random number
Public Declare Sub Randomise Lib "mslib145.dll" (Optional ByVal Seed As Integer)
Public Declare Function Random Lib "mslib145.dll" (ByVal Lo As Long, ByVal Hi As Long) As Long
'Public Declare Function Random Lib "mslib145.dll" (ByVal Lo As Integer, _

```

```

' ByVal Hi As Integer) As Long
Public Declare Function RandomStr Lib "mslib145.dll" (ByVal Length As Long, _
Optional ByVal Style As Integer = RandomStringMixedCase) As String
Public Declare Function EncryptString Lib "mslib145.dll" (ByVal s As String, _
ByVal length As Long, ByVal Password As String, Optional ByVal ExtraSecure As Boolean = True, _
Optional ByVal Salt As String) As Boolean

' Console colours as per "C" conventions //////////////////////////////////////
' Could call these DARK* and the LIGHT* versions just * since the DARK* are very dark
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 ' Yep, American spelling
Public Const FOREGROUND_GREY = 7 ' UK (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

'Console functions
Public Declare Function OpenConsole Lib "mslib145" () As Long
Public Declare Function InNativeConsole Lib "mslib145" () As Boolean
Public Declare Function ConsoleOpen Lib "mslib145" () As Integer
Public Declare Sub CloseConsole Lib "mslib145" ()
Public Declare Sub SetConsoleAttributes Lib "mslib145" (ByVal Foreground As Integer, _
ByVal Background As Integer)
Public Declare Sub ClearConsoleAttributes Lib "mslib145" ()
Public Declare Function HideConsole Lib "mslib145" () As Long
Public Declare Function ShowConsole Lib "mslib145" () As Long
Public Declare Function ConsoleTitle Lib "mslib145" (ByVal s As String) As Long
Public Declare Function WriteLn Lib "mslib145" (Optional ByVal s As String = "") As Long
'We can use WriteLn as a command to write a CRLF
Public Declare Function Writes Lib "mslib145" (ByVal s As String) As Long
Public Declare Function ReadLn Lib "mslib145" (ByVal l As Long) As String
Public Declare Sub Pause Lib "mslib145" (Optional ByVal s As String = "Press a key...")
Public Declare Sub Cls Lib "mslib145" ()
Public Declare Function WhereX Lib "mslib145" () As Integer
Public Declare Function WhereY Lib "mslib145" () As Integer
Public Declare Sub GotoXY Lib "mslib145" (ByVal x As Integer, ByVal y As Integer)
Public Declare Function InKey Lib "mslib145" () As Integer
Public Declare Function GetArgs Lib "mslib145.dll" (ByVal s As String, _
ByRef v As Variant) As Integer

'Process info
Public Declare Function GetPID Lib "mslib145.dll" () As Long
Public Declare Function GetProcessMemoryUsed Lib "mslib145.dll" (ByVal PID As Long) As Long

'MAPI Email
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

'Compression
Public Declare Function RLECompressByteCount Lib "mslib145.dll" (ByVal sInput As String, _
ByVal length As Long) As Long
Public Declare Function RLEUncompressByteCount Lib "mslib145.dll" (ByVal sInput As String, _
ByVal length As Long) As Long
Public Declare Function RLEUncompress Lib "mslib145.dll" (ByVal sInput As String, _
ByVal length As Long, Optional ByRef NewLength As Long) As String
Public Declare Function RLECompress Lib "mslib145.dll" (ByVal sInput As String, _
ByVal length As Long, Optional ByRef NewLength As Long) As String

```

Appendix 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.

Bug Reports and Erratum

Please send bug reports or erratum on this documentation via the website at <http://vbtoolbox.amadis.sytes.net>

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.

Appendix III - Useful References and Links

VBToolbox website

<http://vbtoolbox.amadis.sytes.net>

VBToolbox alternate URL

<http://software.amadis.sytes.net/vbtoolbox>

Visual BASIC Discussion Forum

<http://www.vbforums.com/>

Visual BASIC Express Edition (Free)

<http://www.microsoft.com/express/vb/>

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.amadis.sytes.net>

UPX A Free File-Compression Utility

<http://upx.sourceforge.net>

EditPad – A Free RTF Wordprocessor with encryption provided via VBToolbox

<http://editpad.amadis.sytes.net>

Free Software Page – More free software

<http://software.amadis.sytes.net>

Index

About the Library	2
General Notes On Using the Library	3
Description.....	3
Licensing and Terms of Use.....	3
Declares (Include File - Mslib145.BAS).....	3
Caution - UPX Compression.....	3
Caution - Thread Safety.....	4
Caution - DLLs Are Case Specific!.....	4
Caution - Function Parameters – Use Function Return Values.....	4
Caution - Visual BASIC and Unsigned Long Values.....	4
Caution - String Parameters - Use ByRef in Declares where specified.....	5
Is this a 16 or 32-bit Library?.....	5
DLL Functions Which Return String Values.....	5
Name Conflicts When Calling DLLs.....	5
Correct DLL Function Calling Conventions and Visual BASIC IDE Issues.....	6
Visual BASIC, Windows XP and Data Execution Prevention (DEP) Issues.....	7
DEP Problem Workaround.....	7
Console Functionality.....	8
Server CGI Applications.....	8
Intended Language and O/S Platforms	8
Other DLL Libraries You Can Use.....	8
Why do you call it “BASIC” instead of “Basic”	8
Bug Reporting.....	8
Website.....	8
VBToolbox Installation	9
Installation Procedure.....	9
Installation Troubleshooting.....	9
Visual BASIC – Application Setup Wizard – Install/Setup - Troubleshooting ...	9
Function Interface List	10
DLL Management Functions	10
Function - LibVersion.....	10
String Handling Functions	11
Function - ArgFound.....	11
Function - ArgVal.....	11
Function - CommaStr.....	12

Function - Comma.....	12
Function - CreateGUID.....	12
Function - FillString.....	12
Command - GetArgs.....	13
Function - GetArrayDimensions.....	13
Function - InChrRev.....	13
Function - InChr.....	14
Function - InstrL.....	14
Function - InstrRev.....	14
Function - IsAllChar.....	14
Function - Replace (VB5 Only).....	15
Function - ReplaceChar.....	15
Function - StripLStr.....	16
Function - StripL.....	16
Function - StripRStr.....	16
Function - StripR.....	16
Function - StrRev.....	16
Function - Tokenise.....	17
Function - WildcardMatch.....	18
Arithmetic and Number Functions.....	19
Function - Gcd.....	19
Function - Max.....	19
Function - Min.....	19
Function - PiStr.....	19
Function - Random.....	20
Function - Randomise.....	20
Function - RandomStr.....	20
Function - Integral.....	20
Function - Fraction.....	21
Function - Round.....	21
Date and Time Functions.....	22
Function - UKToISODate.....	22
Function - ISOToUKDate.....	22
Function - UKShortToISODate.....	23
Function - IsLeapYear.....	23
Function - NumOrd.....	23
Function - PHPDate.....	24

Function - PHPDateNow.....	24
Function - DSTAdjust.....	25
Table - PHPDate and – PHPDateNow Token Characters.....	26
Function - VBDate.....	28
Function - VBDateMsecs.....	28
Function - VBDateToCTime.....	28
Function - DateToHex.....	29
Function - HexToDate.....	29
Legacy BASIC Conversion Functions.....	30
Functions - Mki and Cvi (Integer).....	30
Functions - Mkl and Cvl (Long).....	30
Functions - Mkf and Cvf (Float).....	31
Functions - Mkd and Cvd (Double).....	31
Data Encoding Functions.....	32
Function - BaseConv.....	32
Function - BaseConvDouble.....	32
VB Wrapper - Base\$().....	33
Function - Bin8, Bin16, Bin32.....	34
VB Wrapper - Bin\$().....	34
Function - BinToDec.....	34
Function - RotateByte.....	35
Function - RotateInt.....	35
Function - RotateLong.....	35
Function - StrToHex.....	36
Function - StringToHex.....	36
Function - HexToStr.....	36
Function - HexToChar.....	37
Function - HexToInt.....	37
Function - HexToLong.....	37
Function - HexToDouble.....	38
Function - CharToHex.....	38
Function - IntToHex.....	39
Function - LongToHex.....	39
Function - BitPack.....	40
Function - BitUnPack.....	41
Function - EncodeString64.....	41
Function - DecodeString64.....	41

Function - EncryptString.....	42
File and Disk Handling Functions.....	44
Function - DiskFree.....	44
Functions - DiskFreeMeg and DiskFreeGig.....	44
Function - DirExists.....	44
Function - DriveExists.....	44
Function - FileExists.....	45
Function - FilenameMatch.....	45
Function - GetDiskSize.....	45
Function - GetDiskSizeGb.....	46
Function - GetDiskSizeMb.....	46
Function - GetDiskType.....	46
Function - IsCDROMDisk.....	47
Function - IsHardDisk.....	47
Function - IsNetworkDisk.....	47
Function - IsRAMDisk.....	47
Function - IsRemovableDisk.....	47
Function - IsValidDisk.....	47
Function - ListFiles.....	48
Function - ReadFile.....	48
Internet and Network Related Functions.....	49
Function - GetCGIArgs.....	49
Function - IPToLong.....	49
Function - IPMatch.....	50
Function - LongToIP.....	50
Function - MapNetworkDrive.....	51
Function - MapNextFreeNetworkDrive.....	52
Function - UnmapNetworkDrive.....	53
Function - MatchCIDR.....	53
Function - URLEncode.....	53
Function - URLDecode.....	53
Console Functions.....	54
Command - ClearConsoleAttributes.....	54
Command - CloseConsole.....	54
Command - Cls.....	54
Function - ConsoleOpen.....	54
Function - ConsoleTitle.....	54

Command - GotoXY.....	54
Function - InKey.....	55
Function - InNativeConsole.....	55
Function - OpenConsole.....	55
Command - Pause.....	55
Function - ReadLn.....	56
Command - SetConsoleAttributes.....	56
Function - WhereX.....	56
Function - WhereY.....	56
Function - WriteLn.....	56
Function - Writes.....	56
Console Colour Constants.....	57
Native Console App Conversion.....	58
Windows API-Related Functions.....	59
Function - AddEventSource.....	59
Function - AddTrailingSlash.....	59
Function - FileType.....	60
Function - GetDLLFileName.....	60
Function - GetNormalisedPath.....	60
Function - GetOpenFile.....	60
Function - GetSaveFile.....	61
Function - GetSystemDir.....	61
Function - GetWallpaper.....	61
Function - GetWallpaperStyle.....	62
Function - GetWindowsDir.....	62
Function - IsEventSource.....	62
Function - LogEvent.....	63
Function - RemoveEventSource.....	65
Function - ShellRun.....	65
Function - SetWallpaper.....	66
Function - WinSleep.....	66
Function - WindowsSubVersion.....	66
Function - WindowsVersion.....	66
Windows Registry-Related Functions.....	67
Windows Registry Constants.....	67
Function - ReadDWORDFromRegistry.....	67
Function - ReadStringFromRegistry.....	67

Function - WriteDWORDToRegistry.....	68
Function - WriteStringToRegistry.....	68
Windows Process Functions.....	69
Function - GetPID.....	69
Function - GetProcessMemoryUsed.....	69
Graphics Functions.....	70
Function - JPEGCheck.....	70
Function - JPEGHeader.....	70
MAPI and Email Functions.....	71
Function - MAPISend.....	71
Compression Functions.....	72
Function - RLECompress.....	72
Function - RLEUncompress.....	72
Function - RLECompressByteCount.....	72
Function - RLEUncompressByteCount.....	72
Visual BASIC Wrapper Code.....	73
Function - DLLVersion.....	73
Function - IsDLLInstalled.....	73
Visual BASIC Wrapper Code.....	74
Function - Base\$.....	74
Function - Bin\$.....	74
Function - VBStr.....	74
Visual BASIC Wrapper Code.....	75
Function - StripTerminator.....	75
Appendix I - Full VBToolbox Visual BASIC Declares List.....	76
Appendix II - Erratum and Known or Reported Bugs.....	80
Appendix III - Useful References and Links.....	80

This page is intentionally left blank

This document was produced using OpenOffice 3 - <http://www.openoffice.org>
Please support OpenOffice and consider using it in preference to Microsoft Office

