WGLIB version 1.01
Released: July, 1992
Copyright InfoSoft, 1989-1991, 1992

The files that should be included on this disk or .ZIP archive are:
WGLIB101.DOC - Library documentation.
WGLIBGLO.BAS - Compleat declarations for WGLib 1.01.
WGLIBDEM - Various VB source files that make up a demo of WGLib controls and routines
WGLIB.DLL - Windows 3.x Dynamic Link Library for VB development
MAILER - Mailer to register your copy of WGLib.

I. WGLib Compatibility

WGLib 1.01 is a completely new product designed to enhance the functionality and esthetics of Microsoft Visual Basic. The WGLib routines are based on the functionality that GLib 2.01/x brings to BASIC PDS. WGLib is designed to work EXCLUSIVELY with Microsoft's VISUAL BASIC 1.00 and Windows 3.0 / 3.1. Future Microsoft versions of these products may or may not introduce compatibility issues not addressed in WGLIB 1.01.

Unless otherwise noted, all routines used in WGLib are written in assembler with MASM 5.0 or 6.0, with custom controls written in Microsoft C v. 6.00. Except as noted in specific situations we make use of fully documeted DOS, WIN API and VB API calls in order to maintain system integrity under Windows.

II. WGLib 1.01 System Requirements

WGLib is an add-on set of routines for Microsoft Visual BASIC. These routines provide added functionality and power to VB and replace functions lost in the conversion from QB or QBX to VB.

In general, if you can run Visual Basic, you can use WGLIB. Specifically, however, you need at least an 80286 processor (weird to require an 'AT' to run a BASIC huh?), 1 MB of memory, Visual BASIC 1.00 and Windows 3.00. These are largely requirements for VB and/or Windows.

Copyright (C) InfoSoft, 1991, 1992

III. License, Terms and Use:

You are granted free and unlimited personal use of any and all routines in the distribution development library (".DLL") that you find of value. Furthermore, you are free to pass along the BBS distribution files (listed at the start of this document), and only the distribution files, as long as they are passed along as a whole according to the guidlines listed above.

No one is granted any permission to share or pass along any development DLL libraries, object modules or object libraries (LIB). As distributed, the WGLib documentation, demo and the environment library (".DLL") provide you with everything you need to call and execute WGLib routines from within the VB environment.

There are no time limits on how long you may use and examine the viability of the distribution DLL. Likewise, your personal use of the distribution DLL also has no time restriction on it. This provides a generous forum for purposes of sampling, testing and evaluation and allows virtually unlimited latitude in terms of personal use or as a tutorial regarding some of the more advanced features in today's BASIC. This personal use specifically does excludes the right to distribute the .DLL as a runtime module. That is, if your sole intent with WGLib is personal use or experimental use from within the environment, no monies are requested, expected or solicited.

The development time DLL is unsuitable for use as a runtime support DLL and any attempt to use it as such is a violation of these terms. If you find some of the routines of value and desire to incorporate them into .EXE applications, the runtime DLL library of routines and permission to use them in such applications may be licensed as described below. That is, licensing provides both the right to use the Design time DLL, as well the right to distribute the Runtime DLL in support of your application.

Under no circumstances may these routines contained in the user libraries (.LIBs and/or .DLLs) be distributed individually or without the accompanying documentation, which is an integral part of the package, nor may the documentation be altered in anyway.

A RUNTIME DLL and license to use and distribute the same, is granted only to the person or company providing payment or named on the mailer. No grandfather, parent or child usage rights are implied or granted. That is, a RUNTIME license purchased by an individual grant the usage of the library by that individual, not their employer or anyone outside the immediate family. Conversely, a license purchased by a company, corporation, business or other organization does not imply or grant any person employed, allied or associated with that organization personal usage or distribution rights without also purchasing a RUNTIME license.

Copyright (C) InfoSoft, 1991, 1992

Under no circumstances may the routines in the WGLIB package, source, object files, libraries or documentation be distributed by or otherwise become a part of, or affiliated with any group or organization involved in the distribution of what is generally been come to be known as SHAREWARE. WGLib is not SHAREWARE and distribution for disk or distribution fees, or for fees of any sort is expressly forbidden without written consent. This includes supplying the routines, library and or documentation for so-called disk fees.

Finally, the publishers make no claims that the routines herein will fit your needs, simply that in all testing and prior use that they worked for us and that you may find them interesting and helpful in your programming. Any liability for the use, misuse or inability to use the WGLIB routines or libraries, is solely that of the users.

III. Support

As long as it exists, we will support and entertain questions regarding QB and/or GLib as well as VB and/or WGlib via the QuickBASIC conference on The Information Booth, (316) 684 8744, 1200 - 2400 bps, 24 hrs.

If you have a problem with WGLib, you _MUST_ be prepared to supply supporting source code demonstrating the problem you are having. We are more than a little interested in any WGLib or QB/VB bugs that you might find, but do not have the time to track down problems based on vague descriptions of problems when we cannot see if you are using the routine right. PLEASE be specific and supply source examples.

VI Purchasing WGlib

To register this version of WGLib simply read and fill out the enclosed order form (MAILER), enclosing sufficient funds (checks and money orders must be payable in US Funds) and mail it to the address on the mailer. In return you'll receive, by return mail a diskette containing:

- WGLIB101.DOC - Complete documentation
- WGLIB.DLL - Development DLL
- WGLIBR.DLL - Runtime DLL
- WGLIBQRF.EXE - WGLib QRF (quick reference program)
- WGLIBDEM.EXE - Compiled Demo and complete source code

- o SysGauge.EXE - Standalone WIN System Resource gauge
- o WGLIB.HLP - WGLib help in WinHelp format
- o WGLIB101.BAS - Ancillary routines for WGLib 1.01
- o LAUNCHVB.EXE - Program to start up VB with WGLib and it's controls instantly available
- o WGLIBGLO.BAS - Declarations for all WGLib routines
- o WSETUP.EXE - Setup program to install WGLib as well as create Program Manager WGLib group and group items

Copyright (C) InfoSoft, 1991, 1992

Note: Object modules are not supplied since recombining into a DLL requires the use of the SDK and CDK.

Upgrading a previous GLib version must also be done via the mail order form.

In all cases, please the mailer so that we can identify diskette type, VB version etc. Please allow 2-3 weeks for delivery.

Mailing Address:

InfoSoft
WGLib 1.01
PO Box 782057
Wichita, Ks
67278-2057

Copyright (C) InfoSoft, 1991, 1992

Calling Conventions

Each generation in the QB/QBX/VB family seems to produce new, and theoretically improved, variable passing capabilities between the main program and called routines. Along with these capabilities come certain cautions for the programmer.

Visual Basic passes all variables by 32-bit reference by default. This is the QB equivalent of all routines being SEG type calls. The reason for this is due to how the Windows API manages memory and it is not altogether bad. In this format VB passes the segment:offset address of the variable to the called routine. However, this is sometimes overkill: if the routine does not need to actually change a given variable, rather the variable is used as a flag to indicate an operation or such, there is no need to have access to the variable, just the value.

For example, take BitChkInt [which checks the given bit in an integer variable and returns zero or non zero indicating set or not set;
syntax: retc = BitChkInt%(varY%, bitX%)].
Since we are simply going to CHECK the bit x in the variable y, and the return code will indicate the result, the routine need not have access to either item, just the VALUE of them.

To accomplish this, we can use the BYVAL operator to tell VB to simply pass the VALUE of varY and bitX. Rather than passing 4 bytes per parameter, VB will simply pass the value of the integers or 2 bytes each. In so doing, code size is slightly reduced as is execution time (there are fewer ticks used in fetching a passed value versus far addresses).

The down side is that should you pass a variable by far reference when

the routine expects it BYVAL, you will very likely cause a UAE (Unexpected Application Error). This is very unlikely to happen if your DECLARE statements are correct, and the Win API seems to watch all this quite carefully and when the slightest thing goes wrong your program is aborted lest it interfere with other processes running. This is one of the legacies of running in protected mode. So, if you get UAE's try to track down where it happens by using the VB step mode then wherever it crashes, check that the syntax is correct.

One of the most major changes is in string management. By default, when a string is past to a library routine, only a handle to that string is past. To actually access the string data, the DLL routine must call upon the VB API to get a (far) pointer to the string. This would be bad except that VB also allows you to pass a pointer to a string as opposed to a handle. These are declared as BYVAL as well (odd name for that,huh?)

The VB API instructions are a little vague on this but it appears that when passing a string BYVAL, rather than passing a far pointer to a handle, a pointer to the string is passed AND the string is a 'C' type string in that it is null-terminated or ASCII/z. This means that where GLib for BASIC PDS or QB had to copy that string to local memory and tack on a null for DOS file access, we can let VB do the work simply by passing the string BYVAL. This has the side effect of making most VB /

Copyright (C) InfoSoft, 1991, 1992

DOS file functions contain less code than the same routine would for QBX/QB. It also opens up some unique capabilities in some of the file routines.

In these cases, all you need to make sure is that the DECLARE statement is properly set up. For example:

```
Declare Function FOpen% Lib "WGlib.DLL" (ByVal Fil$, FHandle%)
```

When called, VB will make an ASCII/z copy of Fil$ and pass it to the FOpen routine in the WGLIB.DLL. Since the DOS funtion to open a file also requires a null terminated string, the routine receives the string ready to use!

Note that in addition to returning an error code, FOpen will set FHandle to a new value: the handle of the file. In this case, you cannot pass ByVal. First, if you pass a 0 (the VALUE of FHandle), rather than the address of a variable that happens to equal zero, how can the routine change the value of 0 to 5 to indicate the file handle? Second, the routine expects to receive a 32 bit far pointer, not a 16 bit value, so passing FHandle ByVal will likely cause a UAE.

The easiest way to be sure that your code has the proper declaration is

to copy those you need or use from the enclosed WGLIBDEC.BAS into your code. It is not advisable that you load the file into each project as the memory consumed by the file is almost always more than necessary.

Most of the routines in WGLib are FUNCTIONS meaning that they return a value indicating something, as opposed to only changing the values of passed parameters. A few routines are Sub routines. If properly Declared, you can invoke these in your code without the keyword CALL. As far as the documentation goes, we prefer to use CALL in the syntax examples simply to clarify it. You can choose to either use or omit CALLin your invocations, but beware that if you omit CALL, you MUST also omit the parenthesis. For example 'CALL ThatThing(parm1)' becomes 'ThatThing parm1'. Mixing conventions is a recipe for disaster because parentheses around a variable means the same thing as ByVal. So, the syntax 'ThatThing (parm1)' would pass the 16 bit VALUE Parm1 as opposed to the 32 bit ADDRESS of variable Parm1. Keep this in mind as you invoke SUB type routines.

Copyright (C) InfoSoft, 1991, 1992

DLLs

If you are familiar with QLBs for QBX or QB, you will like DLLs much better than LIBs for the most part. For years, I have been preaching and teaching people to make project-specific QLBs so that only as much memory as is required for called routines is consumed. Dynamic Link Libraries combine the best of both QLBs and LIBs.

Unlike DOS's LINK which will pull those routines needed from a LIB into the end EXE, routines are only loaded from DLL's when called (NOT the entire library as with a QLB!). Yet like QLBs, the procedures in a DLL are available at development time.

Another thing you can do with a DLL that you cannot do with LIB/QLBs, is to use multiple DLLs. Since only those routines called are loaded (WHEN they are needed) into memory, you can DECLARE routines from any of a number of DLLs in the same program. However since you cannot easily

create custom DLL's (this requires the Windows SDK and CDK), you would have to distribute each and every DLL referenced including VBRUN100.DLL even if for only 1 routine -- and they can get quite large.

WINDOWS Programming Considerations

A. Windows and VB API considerations

Under Windows, some things get almost hopelessly convoluted in attempting the simplest things. For example, to open a file you no longer simply call DOS but now have to perform some setup steps, allocate a local buffer, then ask the WIN API to ask DOS to open the file and please, give me the handle.

Since Windows is not nearly as mature as DOS, we have avoided using undocumented procedures or calls and adhered to the WIN API guidlines as best we could. Naturally, all video related routines and most BIOS routines have either been removed or reworked to go thru the WIN API.

One major consideration when writing a VB Windows program, and that has to do with files and file I/O. Under QB/QBX and DOS, you could be quite certain that yours what the only program running that might need access or perform I/O on a given file.

Under WINDOWS, you assume the opposite. The first thing that this entails is that you open a file when you need it and not before, and close it as soon as you can. One reason for this is so that you do not inadvertantly deny file access to another process.

Another reason for this is in system resources such as file handles. Opening a file and keeping it open needlessly, denies access to this resource until you release it. Such actions may cause another program or process to abort or crash.

Copyright (C) InfoSoft, 1991, 1992

For these reasons a few of the GLib file functions have been modified for inclusion into WGLib. Primarily, this means that ancillary file functions such as FAttrGet now act on a file name rather than a handle thereby aiding you in keeping files closed except during actual I/O.

B. Calling Convention

Most GLib routines pass parameters by reference - only a few were BYVAL. Since there is actually an advantage to using VB's ByVal method, especially with strings, in converting to WGLib anytime code or execution overhead could be avoided by using alternate syntax or calling conventions, we availed ourselves of this.

This means that most syntax that you may be familiar with has changed. For the most part, all that is required is a modified DECLARE statement.

C. Multiple names

For years we have supported multiple names for some routines so that the programmer could use the one that makes most sense to them. FRep could also be callled as FReplicate, for instance. While such multiple names could be affected for VB as well, we have eliminated them for routine management purposes. You can use the ALIAS parameter to set the name to whatever you like:

```
Declare Function FReplicate Lib "WGLIB.DLL" Alias "FRep" (BYVal _
        Src$, ByVal Dst$, ByVal Buffer$).
```

D. Memory Allocation

Since the Windows API allows greater freedon in allocating memory for local routine use (such as is needed in file copy type operations), we have relieved you, the programmer, of the chore of buffer management.

Examples:

```
DOS GLib: Errc = FCopy(Src$, Dst$, Buffer$)
WIN WGLib: Errc = FCopy(Src$, Dst$)
```

By eliminating the Buffer$, the routine is slightly easier to use, less cumbersome and less error prone. Each routine that requires internal scratch space or a buffer allocates it's own via the WIN API.

Copyright (C) InfoSoft, 1991, 1992

I. WGLib 1.01 Functions and Sub Programs

Name: ArgCnt Type: FUNCTION

Syntax: QArgs = ArgCnt()

Returns the number of arguments in the command tail delimited by a space. That is, with 'FOOBAR.EXE /qwerty /1 /2 /3 /4', ArgCnt would return 5 as the number of command tail arguments. Example and see also: See ArgVar$

Name: ArgVar Type: FUNCTION
Syntax: var$ = ArgVar$(arg)

ArgVar is the complementary routine to ArgCnt(qv) returning a specified argument from the command tail in the programs PSP. ArgVar returns a string variable representing the unparsed specified argument from the command tail (care should be taken that the argument request - arg - is greater than zero).

As with ArgCnt, ArgVar deals with the ACTUAL command tail found in the program's PSP, so accomodations should be made for when your program running as a VB file versus as a EXE (see VBLoaded).
Example - Load command tail elements into an array:

```
QArgs = ArgCnt
REDIM Args$(ArgCnt)
FOR x = 1 to QArgs
   Args$(x) = ArgVar$(x)
NEXT x
```

Name: ArrayIncrI Type: Sub
Syntax: CALL ArrayIncrI(Array%(begin), ByVal Amt%, ByVal NumEls%)

Increments each element in an integer array by a given amount. The array is passed with the starting element as the subscript, the ending point is designated the number of elements to do in NumEls. The constant amount that each element is incremented is that of Amt. This is actually a SUB but can be declared as a function for uniform calling. In any case, the return can be ignored. See Also ArrayTotI and ArrayNDX.

Name: ArrayIncrL Type: Sub
Syntax: CALL ArrayIncrL(Array&(begin), ByVal Amt%, ByVal NumEls%)

This routine performs the same function as ArrayIncrL except that it acts on a long integer array.

Copyright (C) InfoSoft, 1991, 1992

Name: ArrayInitI Type: Sub

Syntax: CALL ArrayInitI(Arry%(begin), ByVal iVal%,_
ByVal NumEls%)

Initializes all or part of an integer array to constant value. All elements in the array beginning with the subscript passed and thru the number of elements indicated by the NumEls parameter are initialized by the value of the iVal parameter. See also ArrayNDX, ArrayIncrI and ArrayTotI.

Name: ArrayInitL Type: Sub
Syntax: retc = ArrayInitL(Array&(begin), ByVal lVal&,_
ByVal NumEls&)

Initializes the desired elements of a long integer array to a desired constant value. Aside from the array being long rather integer and the initialization value being long as well, this is identical to ArrayInitI.

Name: ArrayInitNDX Type: SUB
Syntax: CALL ArrayNDX(Array(begin), ByVal StartVal,_
ByVal NumEls)

Initializes an integer array in sequential fashion (Array(1)=1, Array(2)=2 etc) making it ideal for use in indexing routines. The first element to be acted upon is passed as the Array subscript on the call, and the number of subsequent elements to initialize is controlled by the value of NumEls. See also ArrayInitI, ArrayTotI.

Name: ArrayTotI Type: FUNTION
Syntax: res& = ArrayTotI(Array(begin), ByVal NumEls%)

Scans a desired range of an integer array and returns a long integer representing the sum of all those elements. The start range is passed as the array subscript and the number of elements to sum is passed in the NumEls parameter. Note that while this acts on an integer array, it returns a long integer to avoid overflow possibilities.

Name: ArrayTotL Type: FUNCTION
Syntax: sum& = ArrayTotL(Array&(begin), ByVal NumEls)

This performs identically to ArrayTotI except that it sums a long integer array.

Copyright (C) InfoSoft, 1991, 1992

Name: ArrayToCombo Type: SUB
Syntax: CALL ArrayToCombo(CBCtl As Control, Array$(start),_
ByVal NumEls)

Transfers the contents of an array to a Combo Box.

Doing this in a loop with the Additem method has two drawbacks. First, it is dreadfully slow; some systems experience a 400% speed up of this operation using ArrayToCB. Second, in such a loop, after each item is transferred to the control, VB and/or the WIN API will flicker the screen - whether or not the addition is visible on the screen!

Note: Especially during development, it is important that you periodically use CBClearList to clear the contents of the Combo Box. Each call to ArrayToLB will simply add elements to the control, if they are never cleared you run the risk of running out of memory (VB handles only arrays under 64k - shades of QB 3). LBClearList will quickly clear the contents of a the List Box freeing memory for the rest of your program.
See also: ComboToArray

Name: ArrayToList Type: SUB
Syntax: CALL ArrayToList(Ctl As Control, Array$(start),_
Byval NumEls)

Transfers the contents of an array to a List Box.

Doing this in a loop with teh Additem method has two drawbacks. First, it is dreadfully slow; some systems experience a 300% speed up of this operation using ArrayToLB. Second, in such a loop, after each item is trasferred to the control, VB and/or the WIN API will flicker the screen - whether or not the addition is visible on the screen!

Note: Especially during development, it is important that you periodically use LBClearList to clear the contents of the List Box. Each call to ArrayToLB will simply add elements to the List Box, if they are never cleared you run the risk of running out of memory (VB handles only arrays under 64k - shades of QB 3). LBClearList will quickly clear the contents of a the List Box freeing memory for the rest of your program.
See also: ListToArray

Name: ASCII Type: FUNCTION
Syntax: ret = ASCII(c$)

Returns the ASCII value of a character (or the first character of a string) very much like VB's ASC will but without fear of error when called with an uninitialized string.

Copyright (C) InfoSoft, 1991, 1992

Name: BitChkInt Type: FUNCTION
Syntax: result = BitChkInt%(ByVal value%, ByVal BitNo%)

Tests a specific bit in a passed 16-bit integer value. If the bit is set, the return is non-zero; if not, the return is zero. These Bit????? functions can be handy in creating compact data fields. For example, an integer is more than adequate to store the days of the week which an employee worked. See also BitClrInt, BitSetInt and BitChkLng.
Example:

```
' Bit # 0    1    2    3    4    5    6    7    8....
' Day   Mon   Tue   Wed   Thr   Fri   Sat   Sun

IF BitChkInt(WorkField, 4) THEN
   PRINT "Employee worked Wednesday!"
End if
```

Name: BitClrInt Type: FUNCTION
Syntax: result = BitClrInt%(ByVal value%, ByVal BitNo%)

Clears (sets to zero) a specific bit in a passed 16-bit integer value. See also BitChkInt, BitSetInt and BitChkLng.

Name: BitSetInt Type: FUNCTION
Syntax: result = BitSetInt%(ByVal value%, ByVal BitNo%)

Sets a specific bit in a passed 16-bit integer value. See also BitChkInt, BitClrInt and BitChkLng.
Example:

```
IF EmpClockedIn THEN
   WorkField = BitSetInt(WorkField, 2)
END IF
```

Name: BitChkLng Type: FUNCTION
Syntax: result = BitChkLng%(ByVal value&, ByVal BitNo%)

Tests a specific bit in a passed 32-bit long integer value. If the bit is set, the return is non-zero; if not the return is zero.
See also BitClrLng, BitSetLng and BitChkInt.

Name: BitClrLng Type: FUNCTION
Syntax: result& = BitClrLng&(ByVal value&, ByVal BitNo%)

Clears (sets to zero) a specific bit in a passed 32-bit long integer value. See also BitChkLng, BitSetLng and BitChkInt.

Copyright (C) InfoSoft, 1991, 1992

Name: BitSetLng Type: FUNCTION
Syntax: result& = BitClrLng&(ByVal value&, ByVal BitNo%)

Sets a specific bit in a passed 32-bit integer value. See also BitChkLng, BitClrLng and BitChkInt.

Name: ByteCombine Type: FUNCTION
Syntax: Word = ByteCombine%(ByVal HiByte%, ByVal LoByte%)

Performs a byte combine operation. Combines the byte values of the passed high and low values into a single word value.

Name: ByteSplitLo Type: FUNCTION
Syntax: LoByte = ByteSplitLo(ByVal iVal%)

Returns byte value of the low order portion of a word (INTEGER) value.

Name: ByteSplitHi Type: FUNCTION
Syntax: HiByte = ByteSplitHi(ByVal iVal%)

Returns byte value of the high order portion of a word (INTEGER) value.

Name: ConcaveCtl Type: SUB
Syntax: CALL ConcaveCtl(VBControl As Control, ByVal Thick)

This is a handy and fast routine to draw lines around the specified control so as to give it a sunken, 3D appearance. The routine will work with any control, and goes to some trouble to get the effect just right.

Since larger controls may require a thicker frame or shaded area, the THICK parameter allows you to specify the thickness of that shadow. In general, smaller controls need a thickness of 1 or 2, while the largest frames will only need a thickness of 5.

Note:
The light and dark borders attempt to replace the normal VB borders, but this is apparently somewhat dependant on the type of dispolay sub sytem and it's capabilities. The same code may replace the original borders on one system and simply highlite or accent them on another. This seems especially true of Picture controls

The top border that VB draws for frames is slightly inside the actual frame-owned area to allow for the text title. Since eliminating the title will not eliminate the problem, we adjust where frame shading is, to locate it outside the VB frame like the Picture control.

Copyright (C) InfoSoft, 1991, 1992

Name ConcaveFrm Type: SUB
Syntax: CALL ConcaveFrm(Frm As Form, ByVal Thick)

Like ConCaveCtl this adds a 3d sunken, concave effect, but to a Form. The amount of shading required by a Form can be much more than a control due to the size, usually a thickness of 5 is sufficient.

Name: ConvexCtl Type: SUB
Syntax: CALL ConvexCtl(VBControl As Control, ByVal Thick)

This handy and fast routine draws lines around the specified control so as to give it a raised, 3D appearance. The routine will work with any control, and goes to some trouble to get the effect just right.

Since larger controls may require a thicker frame or shaded area, the THICK parameter allows you to specify the thickness of that shadow. In general, smaller controls need a thickness of 1 or 2, while the largest frames will only need a thickness of 5. Very, very effective images can be produced using Convex Frames and Forms with Concave controls.

Note: The light and dark borders attempt to replace the normal VB borders, but this is apparently somewhat dependant on the type of dispolay sub sytem and it's capabilities. The same code may replace the original borders on one system and simply highlite or accent them on another. This seems especially true of Picture controls

The top border that VB draws for frames is slightly inside the actual frame-owned area to allow for the text title. Since eliminating the

title will not eliminate the problem, we adjust where frame shading is, to locate it outside the VB frame like the Picture control.

Name ConvexFrm Type: SUB
Syntax: CALL ConvexFrm(Frm As Form, ByVal Thick)

Like ConvexCtl this adds a 3D raised, convex effect to a Form. The amount of shading required for a Form is usually more than needed on a control due to the size, usually a thickness of 5.

Name: DayOfYr Type: FUNCTION
Syntax: DaySoFar = DayOfYr%()

This simply returns the day of year as an integer. It is accurate for dates of 01-01-1980 to 02-28-2???. This is a FUNCTION that works off the current system date, returning the day count in the FUNCTION name.
Example:

```
DECLARE FUNCTION DayOfYr%
..
..
TodayCount = DayOfYr
```

Copyright (C) InfoSoft, 1991, 1992

Name: DeciBin Type: FUNCTION
Syntax: bin$ = DeciBin$(ByVal value%)

Returns a 16 character string representation of the passed value. This is like BASIC's HEX$ except it returns a string of 0's and 1's.
Example:

```
Bin$ = DeciBin$(2)          ' returns "0000000000000010"
```

Name: DOSDate Type: FUNCTION
Syntax: DOW = DOSDate(mo, day, yr)

Return the current system date information as well as the current day of week.

Name: DOSInt Type: FUNCTION
Syntax: CALL DOSInt(Regs AS CPURegs)

One of the things missing in Visual BASIC is access to standard DOS

interrupt calls. DOSInt allows you to continue such direct DOS calls. The structure CPURegs is defined in WGLIB.BAS (the declarations list) as follows:

```
Type CPURegs
    AXReg As Integer
    BXReg As Integer
    CXReg As Integer
    DXReg As Integer

    SIReg As Integer
    DIReg As Integer
    ESReg As Integer
    DSReg As Integer
    Flags As Integer
End Type
```

Upon entry set the registers to their desired values, when invoked DOSInt will call Int 21h (and Int 21h only!), and return the post interrupt register contents.

Note: This is a _very_ advanced routine and you should have a very good understanding of DOS, the WIN API and the VB API before tinkering with it. Any damage resulting from the use or misuse of this routine is soley that of the user.

Copyright (C) InfoSoft, 1991, 1992

Name: DOSTime Type: FUNCTION
Syntax: ret = DOSTime(hrs, mins, secs)

Rather than tearing apart BASIC's TIME$ with MID$ to determine the current hour, minute, second, DOSTime allows instant access to these items in integer format with no string garbage generated.

Name: DosVer Type: FUNCTION
Syntax: OEM = DosVer%(Major%, Minor%)

Returns the DOS Version the system is running under as well as the OEM number, if any) - this is the DOS supplier for example, IBM's PC-DOS will return 00 as the OEM number, DEC as 16h etc. The Major and minor versions will return what function 30h returns correct regardless of the

OEM. Note that this function respects SETVER included with DOS 5.00; see DosVerTrue for the true DOS version.

Name: DosVerTrue Type: FUNCTION
Syntax: retc = DOSVerTrue(Ver%, Rev%, HiLo%)

If DOS 5.00 is installed, this will return the true DOS version whether SETVER is installed or not. The true DOS version is returned as a whole number in Ver (DOS 5.00 will be returned as 500). Rev indicates any revison codes returned, and HiLo indicates whether DOS is loaded High or Low.

Name: DrvFree Type: FUNCTION
Syntax: FreeBytes& = DrvFree&(ByVal Drv%)

Returns the total free space in bytes on the specified drive. Input parameter is drive number to poll: 1= A:, 2=B: etc; 0 = default.
Example:

```
a=0                          ' read default drive
FreeSpace& = DrvFree(a)
```

See also DrvSiz

Name: DrvGet Type: FUNCTION
Syntax: drv = DrvGet

This gets the default disk drive. It returns the ASCII code of the letter to avoid the confusion of drive numbering. Converting to a character is simple with BASIC's CHR$ function.

Copyright (C) InfoSoft, 1991, 1992

Name: DrvSpace Type: FUNCTION
Syntax: TotalBytes& = DrvSpace&(ByVal Drv%)

Returns the total bytes (used and free) on the specified drive. Input parameter is drive number to poll: 1= A:, 2=B: etc; 0 = default.

Example:

```
a=0                          ' read default drive
```

TotalBytes& = DrvSpace(a)
See also DrvFree

Name: DrvSet Type: FUNCTION
Syntax: ret = SetDrv(ByVal drv$)

Sets or resets the default drive. To set the drive, simply pass it the letter, upper or lower case, to log into. This removes the ambiguity of drive numbers (Hmm, is drive 1 A: or B: in this case...). SetDrv returns nothing and if passed a bad parameter, DOS simply rejects it leaving the default drive unchanged. See also DrvGet. Example:

```
errc = DrvSet("d")
```

Name: DrvStat Type: SUB
Syntax: CALL DrvStat(ByVal Drv%, SecClus, ClusAvail%, SecBytes%,_
TotClus%)

Returns key statistics on the drive requested. On entry the first parameter is the drive number to poll: 1= A:, 2=B: etc;
0 = default.

On return, the parameters representing Sectors per Clusters, Clusters Available, Bytes per Sector and Tot Clusters on the drive respectively are filled in with the drive's values.

Name: EmbossFrm Type: SUB
Syntax: CALL EmBossFrm(Frm As Form, ByVal Top% ByVal Left&_
ByVal Bottom, ByVal Right%, ByVal Thk%)

Draws a very petite 3D frame like structure on a form.

Even though it is very effective, there are one or two things about VB frame controls with our ConcaveCtl or ConvexCtl shading that are aesthetically out of tune. One is that the top of the VB control is not where the top line is drawn, and you cannot hide this with a frame style parameter. Another is that on some frames and forms even a petite shading effect is a little overwhelming when added to the standard frame.

EmbossFrm will draw a 3D frame-like structure on the given form at the passed Twip coordinates. The embossed effect makes the framed area appear to stand out (convex) from the form. We find that a thickness of 1 or 2 is perfect, but that is up to you.

Copyright (C) InfoSoft, 1991, 1992

Notes: A) This is NOT a custom control, it is simply an effect to

highlight or draw attention to portions of the frame, or controls located inside the embossed area.

B) The passed coordinates MUST be in twips format, the default ScaleMode for VB forms.

C) The procedure will NOT obliterate or draw over existing controls. This is a true asset as placing a LABEL control in the path of the top line provides for the appearance of a very effective 3D frame.

See also: EmpressFrm

Name: EmpressFrm Type SUB
Syntax: CALL EmpressFrm(Frm As Form, ByVal Top% ByVal Left&_
ByVal Bottom, ByVal Right%, ByVal Thk%)

Draws a very petite 3D frame like structure on a form.

While there is really no such word as 'empress', it evokes images of the inverse of emboss and places EmpressFrm directly after EmbossFrm in alpha listings.

EmpressFrm will draw a 3D frame-like structure on the given form at the passed Twip coordinates. The effect makes the framed area appear to be pressed into (concave) to the form. We find that a thickness of 1 or 2 is perfect, but that is up to you.

Notes: A) This is NOT a custom control, it is simply an effect to highlight or draw attention to portions of the frame, or controls located inside the empressed area.

B) The passed coordinates MUST be in twips format, the default ScaleMode for VB forms.

C) The procedure will NOT obliterate or draw over existing controls. This is a true asset as placing a LABEL control in the path of the top line provides for the appearance of a very effective 3D frame.

See also: EmbossFrm

Copyright (C) InfoSoft, 1991, 1992

Name: EqInfo Type: SUB
Syntax: CALL EqInfo(Ser, Par, Floppy)

Returns some basic hardware configuration information as contained in low memory. Should a CMOS system encounter an error during the POST (Power on Self Test) this information could be incomplete. See also: FloppyType.

Parameters
Ser - Returns number of serial ports installed
Par - Returns number of parallel ports installed
Floppy- Returns 0,1,2 as number of physical floppies installed

Name: FAttrGet / FAttrSet Type: FUNCTION
Syntax: errc = FAttrGet%(ByVal fil$, fattr%)
Syntax: errc = FAttrSet%(ByVal fil$, ByVal fattr%)

These allow access to get or set the file attributes for a given file. The file need not be open as the operations are performed on file names. File attributes are as follows:

00 - Normal 04 - System
01 - Read Only 32 - Archive
02 - Hidden

The archive bit is typically used by back up programs to determine if a file has been backed up since the last write process. To combine attributes, just add the values, ie: Read Only - Hidden would be 3 because 1+2=3.

The function returns an error code indicating any error that DOS encountered trying to execute the function. The most likely error would be error code 6 - File Not Found. Note that when using FAttrSet, DOS will not allow you to set or change volume labels or directories.
Error Codes:
-1 = Invlid File attribute 3 = Path not found
2 = File not found 5 = Access denied

Example:

```
DECLARE FUNCTION FAttrGet%(ByVal fil$, ByVal attrib%)
.
.
fil$ = "myfil.txt"
failure = FAttrGet(fil$, attrib)
IF failure THEN MsgBox("Sorry, cannot find "+fil$, 48, "Error!")
```

Copyright (C) InfoSoft, 1991, 1992

Name: FClose Type: FUNCTION
Syntax: errc = FClose(handle%)

FClose performs the opposite of FOpen, that is, to close out a file handle. FClose does some positive error checking to make sure that you do not attempt to close one of the standard DOS file handles (like the keyboard, monitor etc) and will return a error code of 6 - Invalid File handle. Note that FClose will zero out the variable thus preventing you from accidentally attempting firther file operations on a handle that has been closed!

See also FOpen FUnique, FCreat, FSetPtr, and DOS File Functions.
Example:

```
DECLARE FUNCTION FClose%(ByVal fhandle%)
errc=FOpen(Fil$, FHandle)
.                          ' FHandle == 6
.
result = FClose(FHandle)
.                          ' now FHandle = 0
IF result THEN GOSUB ErrorControl
```

Name: FCopy Type: FUNCTION
Syntax: errc = FCopy(ByVal source$, ByVal dest$)

This function copies a disk file using a buffer supplied by the main program, and performs about as fast as the DOS COPY command. You pass it a source and destination string (paths / drives are ok) representing filenames.

The FUNCTION returns a variety of error conditions:

2 = File Not Found
3 = Path not Found
4 = No Handle ("Too Many Files")
5 = Access Denied

Example:

```
DECLARE FUNCTION FCopy%(ByVal source$, ByVal dest$)
..
result = fcopy("WGLIB.DLL","\WINDOWS\SYSTEM\WGLIB.DLL", SPACE$(4096))

IF result THEN
  MsgBox("Oops!  Error - Check parameters!", 48, "FCopy Error")
```

```
END IF
```

```
    Name: FCount                Type: FUNCTION
Syntax: NumFiles = FCount(ByVal fil$)
```

This is can be an indespensible tool - it quickly returns a count of the number of files matching the given mask. This is extremely useful in determining how large an array should prior to a directory related function. If an invalid path or drive is included in the mask$, no matching files will be found.

Copyright (C) InfoSoft, 1991, 1992


```
    Name: FCreat                Type: FUNCTION
Syntax: errc = FCreat(ByVal fil$,  FHandle%)
```

FCreat is similar to FOpen except that instead of opening an existing file, we are CREATING a NEW file. If the file already exists it is trunacted as if opened FOR OUTPUT in BASIC. As with FOPEN, and all the DOS File Functions, error codes returned in the BASIC FUNCTION format and are:

3 Path not found
4 No handle available ("Too many files")
5 Access denied (returned when attempting to FCreat a file that is already open - use FOpen in this case).

Example:

```
DECLARE FUNCTION FCreat%(ByVal fil$, FHandle%)
.
.
fil$="mainprg.sys": attrib=0
IF fcreat(fil$, fhandle)=0 THEN
   MsgBox("New file, " +fil$+ " successfully created!", 48, "Info")
ELSE
   GOSUB WhatsGoingOn
END IF
```

```
    Name: FDateGet/FDateSet     Type: FUNCTION
Syntax: errc = FDateGet(ByVal fil$, mo%, day%, yr%)
Syntax: errc = FDateSet(ByVal fil$, ByVal mo%, ByVal day%, ByVal_
                yr%)
```

These allow you to SET or GET the date for a file. The file need not be open since the function(s) are accessed by file name. Because of this, it would make sense to use FExist to see that the file is there. The year parameter may be either 2 or 4 digits (ie 1989 or 89). Eg:

```
OPEN "foo.bar" FOR RANDOM AS #1
```

```
handle = FILEATTR(1,2)
errc = FDateSet(handle, 1, 1, 80)        ' set date to 1/1/1980
```

Name: FDelete Type: FUNCTION
Syntax: errc = FDelete(ByVal fil$)

Simply deletes the file indicated by fil$ from the disk. The file should NOT be open. This uses the DOS function UNLINK to simply remove the first character so it may be UNDeleted with any of a number of low level Disk tools. Example:

```
errc = FDelete("foo.bar")
```

Copyright (C) InfoSoft, 1991, 1992

Name: FEOF Type: FUNCTION
Syntax: errc = FEOF(ByVal FHandle)

Sets the file pointer to the end of a file opened via FOPEN. You may want to actually set the pointer to ONE BYTE less than the end of text files to be sure that subsequent FWRITE funtions overwrite any hard EOF marker (ASCII 26) left by many text editors. In this case use a combination of FSIZEh and FSETPTR.

Additionally, FEOF checks for invalid handles as well as the 4 standard DOS handles and aborts to return an error code of 6 - Invalid handle. Using FOPEN and then FEOF is the equivalent of opening a file in APPEND mode using BASIC's intrinsic file functions. See also: FSetPtr.
Example:

```
DECLARE FUNCTION FEOF%(ByVal fhandle%)
.
IF FOpen("longtext.fil", fhandle) THEN
   GOSUB StartNewFile
ELSE
   j= feof(fhandle)
   IF j THEN GOSUB WeirdError
   GOTO AppendToText
END IF
```

Name: FExists Type: FUNCTION
Syntax: ExistCode = FExists(ByVal fil$)

This routine sets the return code (non zero) if the given file exists and no path or other error is encountered. See also FileDNE.
Example:

```
DECLARE FUNCTION FExists%(ByVal fil$)
..
fil$ = "foo.bar"
IF FExists(fil$) THEN
   ret = MsgBox(fil$ + " already exists! Overwrite?",48, "Query")
END IF
```

Name: FFlush Type: FUNCTION
Syntax: errc = FFlush(ByVal FHandle)

This flushes the DOS file buffers forcing any buffered data to disk. If you are using the WGLIB DOS file functions, (FOPEN, FWriteStr, FWriteAry etc), FFlush will dump any buffered data to disk much faster than closing and then reopening the file will. FFlush will NOT dump the buffer for files opened using VB functions - VB does some significant buffering of its own which does not respond to FFlush. Pass FFlush the handle associated with the file, any return indicates an error code the same as the other DOS file functions. Example:

```
errc = FFlush(fhandle)
```

Copyright (C) InfoSoft, 1991, 1992

Name: FileDNE Type: FUNCTION
Syntax: errc = FileDNE(ByVal fil$)

This provides the inverse logic of FExists. Instead of testing if a file DOES exist, it returns non zero if the file DOES NOT exist. This can be handy for logic or statements that work better with the non zero return. See also FExists. Example:

```
'REM Instead of:
IF NOT FExists(Fil$) THEN...   or   IF FExists(fil$) = 0 THEN...

IF FileDNE(fil$) THEN
   ret = MsgBox("File Does Not Exist - Create?", 32 + 3, "Warning!")
END IF
```

Name: FileLoad Type: FUNCTION
Syntax: errc = FileLoad(fil$, size&, Buffer$)

One of the slowest operations in VB is the assignment and manipulation of string space, particularly string space allocation (your VB program has to ask 'C' to ask the WIN API to allocate memory from DOS and back down

the chain to your program). FileLoad will quickly read a file of less than 64k into memory. Upon entry, fill in fil$ with the name (drive and path ok) of the file to read, and initialize Buffer$ to a minimum of 1 space. When called FileLoad will reallocate Buffer$ to the required size, read the file into that string, and return the file's size to you in the size& parameter.

Name: FloppyType Type: FUNCTION
Syntax: retc = FloppyType(ByVal Drv)

Returns a code indicating the type of floppy drive that the designated drive is (default drive = 0, 1= A: ...)

Code	Floppy Type	Code	Floppy
0	Unknown type or no such drive exists		
1	360k	4	1.44 Mb
2	1.2 MB	5	2.88 Mb

Name: FMove Type: FUNCTION
Syntax: errc = FMove(ByVal source$, ByVal dest$)

This works the same as a FCopy/FDelete combination with the source file being deleted after the destination file is created. Any errc return indicates an error such as disk full, file already exists and is Read Only, or disk is write protected. The source and destination file name may be on different devices.
Example:

```
errc = FMove("foo.bar", "A:foo.bak")
```

Copyright (C) InfoSoft, 1991, 1992

Name: FOpen Type: FUNCTION
Syntax: errc = FOpen(ByVal fil$, fhandle%)

FOpen is a standard DOS function (albeit executed thru the Windows API) to open a file via a file handle. The DOS function requires that an open mode or style be indicated, and Windows has further expectations or requirements, so the mode is hard coded to open the file in READ/WRITE mode and your process will have exclusive access to it.

The fhandle parameter returns the file handle and FOPEN returns any error codes DOS or the WIN API encounter. An error can occur if the file is already open, file not found etc. In this case, the errorcode or result will indicate what happened and any fhandle number should be ignored.
Example:

```
DECLARE FUNCTION FOpen%(ByVal fil$, fhandle%)
```

```
..
fil$="myprog.dat"
IF fopen(fil$, fhandle) THEN
   GOSUB FileError
ELSE
   ' fil$ is now open under the handle ID of fhandle
END IF
```

Name: FOpenW Type: FUNCTION
Syntax: errc = FOpenW(ByVal fil$, FHandle)

This works very much like FOpen, but if the file does not exist, the WIN API is set to pop up a dialog box asking the end user to insert a disk with that file on in drive A:. This is a less than elegant way to handle missing files, but can be handy in quick and dirty applications.

Name: FReadArray/FWriteArray Type: FUNCTION
Syntax: errc = FReadArray(SEG arry, ByVal FHandle, BYTES)
Syntax: errc = FWriteArray(SEG arry, ByVal FHandle, BYTES)

FReadArray reads data from a disk file directly into an array. The file must have a valid DOS File Handle which is accomplished either by using FOpen or FILEATTR on a BASIC file number. Elements indicates the number of BYTES to fill (remember that each elements in the array is 2 Bytes), upon return from the function BYTES is reset to the actual number read (in case EOF is encountered before all the requested bytes can be read). ERRC is set to indicate any DOS error encountered.

See the complimentary function FWriteArry.
Example:

Copyright (C) InfoSoft, 1991, 1992


```
DECLARE FUNCTION FReadArry%(SEG arry%, ByVal Fhandle%, Bytes%)
..
..
REDIM Emps(NumEmps) AS StructEMP     ' size array
..
fil = FREEFILE                       ' get next BAS file no
OPEN "screens" FOR RANDOM AS #fil LEN=LEN(EMPs(1))
handle = FILEATTR(fil,2)
Bytes = LEN(Emps(1)) * NumEmps       ' size of record
```

```
errc = FreadArray(Emps(1), handle, Bytes)   ' read all emps into
                                  '   struct
```

Name:FReadByte/FWriteByte Type: FUNCTION
Syntax: errc = FReadByte(ByVal FHandle, byte)
Syntax: errc = FWriteByte(ByVal FHandle, byte)

This is similar to BASIC native BINARY file I/O, allowing you to write or read a single byte to disk. This is not a character but a byte, or the ASCII value of the byte (ASC(ch$)) to be sent or read from disk.

You must have a valid, open handle for the destination file - use FOpen or BASIC's FILEATTR for this. This can be considerably quicker for byte I/O than BASIC's BINARY method. Example read 5 bytes:

```
FOR x = 1 TO 5
   errc = FReadByte(handle, byte)
NEXT x
```

Name: FReadStr/FWriteStr Type: FUNCTION
Syntax: errc = FReadStr(ByVal Bufr$, ByVal Fhandle, CharsCnt)
Syntax: errc = FWriteStr(ByVal Bufr$, ByVal Fhandle, CharsCnt)

FReadStr reads data from a disk file directly into a string variable, FWriteStr writes that string. The file must have a valid DOS File Handle which is accomplished either by using FOpen or FILEATTR on a BASIC file number. CharCnt indicates the number of CHARACTERS or BYTES to read or write. Upon return from the function, CharCnt is reset to the actual number read (in case EOF is encountered before all the requested characters can be read). ERRC is set to indicate any DOS error encountered. Because FReadStr is in high-speed assembler, you must initialize the buffer or string that will hold the read data to at least as long as the number of characters to read or an error will occur. Example:

```
DECLARE FUNCTION FReadStr%(ByVal Buffer$, ByVal Fhandle%, Bytes%)
..
..
message$ = SPACE$(25)
chars = 25
fil = FREEFILE                  ' get next BAS file no
OPEN "myprog.dat" FOR OUTPUT AS #fil
handle = FILEATTR(fil,2)
errc = FReadStr(Message$, handle, chars)
```

Copyright (C) InfoSoft, 1991, 1992

Name: FRecGet/FRecPut Type: FUNCTION

Syntax: errc = FRecGet(ByVal FHandle%, ByVal size%, fStr As Any)
Syntax: errc = FRecPut(ByVal FHandle%, ByVal size%, fStr As Any)

These perform essentially the same function as QB's GET # and PUT # by reading a single record from the current location of the file pointer represented by the handle passed. Use FsetPtr (qv) to set the file pointer to the desired location. These are the single record version of FRecGetA/FRecPutA (qv). In using these over PUT/GET, you can completely bypass QB's file I/O, buffering and obnoxious error routines and interpet the return code for errors. The biggest advantage to using these will be to those who are also using the multiple record I/O routines (FGet/PutRecA). Note that while these are setup for TYPEd file I/O, they will work for FIELDed files, though the hassle involved with this is probably not worth the effort. An error return indicates failure, generally an invalid handle was passed (6).

See also FGetRecA/FSetRecA, FSetPtr.

Name: FRecGetA/FRecPutA Type: FUNCTION
Syntax: errc = FRecGetA(ByVal FHandle, ByVal Quan, ByVal size,_
fStruct As Any)
Syntax: errc = FRecPutA(ByVal FHandle%, ByVal Quan%, ByVal size%,_
fStruct As Any)

These perform essentially the same function as QB's GET and PUT, with the exciting difference that rather than single record operations, an entire TYPE array of records (designated by Quan) are read or written at once. Rather than reading (or writing) many records from within a QB FOR...NEXT loop, you can read as many as desired in one pass and much quicker, making it ideal for large database operations:

```
errc = FGetRecA(handle, 128, LEN(Emp(1)), Emp(1))
```

This would read 128 records from the file handle (starting at the current location) and place them in the TYPE array Emp beginning at subscript 1. The size of the (TYPE) structure is passed so that the functions can calculate the number of bytes to read.

Care must be taken that the read or write (transfer) request does not exceed 64k (Quan recs times Size of struct) since VB does not handle huge arrays at this time. The only likely return code placed in errc, generally signals an invalid handle (6) or a SHARE violation (5).

NOTE: Records or data are read from the file at the current DOS file pointer position. Use FSetPtr to locate this to the desired spot: QB's SEEK may not always do the equivalent on random files.
See also FGetRecA/FSetRecA, FSetPtr.

Copyright (C) InfoSoft, 1991, 1992

Name: FRename Type: FUNCTION
Syntax: errc = FRename(ByVal oldf$, ByVal newf$)

This allows you to rename a disk file. Since a simple DOS function is used, both the source and destination files must reside on the same device (you may not move the file from one drive to another with this function - see FMove, FReplicate or FCopy for that). Any return indicates an error such as an attempt to span drives. Example:

```
errc = FRename("foobar.new", "foobar.old")
```

Name: FRep Type: FUNCTION
Syntax: errc = FRep(ByVal source$, ByVal dest$)

This works very similarly to FCopy, except the file date and time are preserved on the destination file. You supply the buffer by simply passing a string or temporary variable. Any errc return indicates an error such as disk full, file already exists and is Read Only, or disk is write protected. Eg:

```
errc = FRep("foo.bar", "A:foo.bak")
```

Name: FSetPtr Type: FUNCTION
Syntax: errc = FSetPtr&(ByVal Fhandle%, ByVal RecNo&,_
ByVal RecSize%)

Moves the DOS file pointer to a specified location in a file (usually to a specific record in a random file) opened with a handle. Note that RecNo is a LONG INTEGER so that very large files can be addressed. This function is used with FGetRec, FPutRec, FGetRecA and FPutRecA (qv) to position the DOS file pointer to a specific location. Note that this acts directly on the DOS file pointer and therefore acts differently than BASIC's native SEEK would (due to significant file buffering performed by VB iteself), and returns a long integer indicating the byte position of the DOS file pointer. Note that the use of this function is a prerequisite to calls to FGetRec, FPutRec, FGetRecA and FPutRecA.

See also FEOF, FRecGet, FRecPut, FRecGetA, FRecPutA.
Example:

```
TYPE struct
   RecNo AS INTEGER
   AName AS STRING * 25
   BStuff AS STRING * 15
   FooBar$ AS STRING * 15
END TYPE
DIM RecThing AS STRUCT

RSize = LEN(RecThing)        ' calc once rather than many LEN calls
```

```
OPEN datfil$ for RANDOM as #f LEN = RSize
Fhandle = FILEATTR(f, 2)      ' Ask BASIC for handle of file
..
..
fpos& = FSetPtr&(FHandle, RecNo&, RSize)   ' seek to RecNo
```

Copyright (C) InfoSoft, 1991, 1992

Name: FSizeH Type: FUNCTION
Syntax: SizeOfFile& = FSize&(ByVal FHandle%)

Passed a valid file handle from FOpen or FILEATTR, this will return a long integer representing the size of the file in bytes. See also: FSizeN

Name: FSizeN Type: FUNCTION
Syntax: SizeOfFile& = FSize&(ByVal Fil$)

Passed a string representing a valid file, this will return a long integer representing the size of the file in bytes. See also: FSizeH

Name: FTimeGet/FTimeSet Type: FUNCTION
Syntax: errc = FTimeGet(ByVal fil$, hr, min, sec)
Syntax: errc = FTimeSet(ByVal fil$, ByVal hr, ByVal min, ByVal sec)

Like the names imply, these allow you to set or get the timestamp of a file or the last time it's directory entry was updated. You pass it the name of the file and the only likely error code is that of File Not Found. Example:

```
 errc = FTimeSet(Fil$, 10, 10, 10)   ' set time to "10:10:10"
```

Name: FUnique Type: FUNCTION
Syntax: errc = FUnique(ByVal fil$, ByVal attr, Fhandle)

This DOS disk file function creates a file with a unique name, which makes it ideal for scatch data files. Rather than HOPING that "mydat.@@@" is a unique filename, FUnique makes SURE that a file is in fact unique. DOS will create such a file in the specified directory and open it with read write access with the attributes you specify and return to you a valid handle for use - or an error code.

FUnique will also return to you the actual file name that is open for

access, but to do so requires very specific handling. First, populate a string variable with the path where you wish the temporary or scratch file to exist. Terminate this string with a NULL (CHR$(0)), then tack on a minimum of 12 spaces. DOS will overwrite the null and most of the spaces with the actual file name.

NOTE: The unique file is OPEN with a handle! Subsequent file access should be thru WGLib file functions, or close the file handle and reopen it using the name and Visual BASIC.
Example:

Copyright (C) InfoSoft, 1991, 1992


```
DECLARE FUNCTION FUnique%(ByVal path$, ByVal attrib%, fhandle%)
..
..
tempfil$ = "C:\BIN\" + CHR$(0) + SPACE$(13)
IF FUnique(fil$, 0, fhandle) THEN       ' path, normal file
   GOSUB InvalidInfo
ELSE
   errc = FClose(Fhandle)
   OPEN TempFil$ For Output as #1
END IF
```

Name: GetWGlibVer Type: FUNCTION
Syntax: WGVer = GetWGlibVer()

WINDOWS programs are unique in that called routines such as those in WGLib remain external to the program even at run time AND the library name is referenced in your VB code (in the DECLARES).

To preclude any possible confusion, this routine will a code indicating the WGLib version. This is returned as a whole number, meaning 1.01 will be returned as 101.

Name: INCR / DECR Type: FUNCTION
Syntax: result = INCR(ByVal x, ByVal y)
Syntax: result = DECR(ByVal x, ByVal y)

Provides an easy method of incrementing or decrementing a VB variable by a given amount. The function return is the result of the operation.
Example:

```
DECLARE FUNCTION Incr%(ByVal x%, ByVal y%)
DECLARE FUNCTION Decr%(ByVal x%, ByVal y%)
..
i = INCR i, 5                   ' same as i=i+5
```

```
j = DECR j, q                    ' same as j=j+q
```

```
    Name: INSTRI                 Type: FUNCTION
  Syntax: result = INSTRI(ByVal Start, ByVal FindIn$, ByVal LookFor$)
```

This function is identical to the native INSTR except that it is case insensitive. That is, INSTRI will return an integer pointing to the first occurrence of either "CD" or "cd" in "ABCDEF" or "abcdef". Like INSTR, INSTRI allows the use of a starting point to begin the search withing the string to BE searched, unlike BASIC, this is not optional. Like BASIC, the return is a relative pointer from the starting point. Further, the routine will recognize "?" in the search string as a match-all wild card. Example:

```
DECLARE FUNCTION INSTRI%(ByVal start%, ByVal a$, ByVal b$)
..
j = INSTRI(1, "AbCdEFGhiJkL", "cDe?")      ' returns 3
```

Copyright (C) InfoSoft, 1991, 1992


```
    Name(s): IsCharxx            Type: FUNCTION (s)
  Syntax: status = IsChar????(ByVal c$)
          status = IsStr?????(s$)
```

The 'izzy' collection provides for assembler level replication of the highly useful IsChar???? macros found in 'C'. Where the 'C' macros work only on a single character, WGLIB provides routines that work on characters (IsChar???) or whole strings (IsStr???). When using a string however, a single non matching character forces a false return. Example:

```
IF IsCharAlph("The quick brown fox is really a lazy dog.") THEN
   Text1.Text = "All Alpha chars!"
ELSE
   Text1.Text= "Non alpha chars in string!"
END IF
```

In this example, the return is FALSE, the string is NOT all alpha characters - the spaces and period in the string cause a zero (FALSE) return. All the IZZY functions return a 0 or -1 based on the function. The IsChar functions are:

IsCharAlph - Test if the character is between A-Z or a-z.

IsCharAlNum - Test for characters A-Z, a-z and 0-9.

IsCharUppr - Tests to see if the character is Upper case. Very useful when you want to preserve the case of some input or want to avoid generating string garbage in reassigning UCASE(x$) to a new, temporary variable.

IsLowr - Similar to IsUpper - tests for lower case.

The IsString functions are:

IsStrAlph - Checks an entire string for alpha components.

IsStrAlNum - Checks that all character entries in a string are Alpha-Numeric.

IsStrText - Similar to AlNum but allows the inclusion of spaces and punctuation (ASCII 32 to 127).

Copyright (C) InfoSoft, 1991, 1992

Name: Julian Type: FUNCTION
Syntax: JDay& = Julian&(ByVal month, ByVal day, ByVal year)

Returns the TRUE julian date for the passed month/day/year. What many dating schemes call "julian" is merely an ordinal day code or an ordinal code serialized with the year such as 90102 which is supposed to indicate the 102nd day of 1990. Even worse, there are some that return a SERIAL date by calculating the number of days since 1/1/1. These are invariably wrong since they do not take into account that leap years are NOT every 4 years (years such as 1700, 1900 and and 2100 are not leap years); also they tend to ignore that there were 11 days supressed in 1582 (you went to bed on Oct 4, you woke up on Oct 15).

A modified form of serial dating that is widely used is to calculate the number of days elapsed since Jan 1, 1900. This modified julian date method is used by LOTUS 1-2-3. This is generally used because invoicing needs and such usually need not extend back to before 1900 and returns a smaller number (on the order of 32,000).

A _TRUE_ julian date such as those returned by Julian&, represent the number of days passed since Jan 1, 4713 BC. Note that Julian& returns a long integer representing this Julian date. Because Julian& will not accept dates before Jan 1, 0001 AD, the smallest number returned is

1721424. That is, Julian does not convert BC dates.

Such a dating method allows for significant, long range date calc-
ulations, such as getting the date for a day x days in the future or x
days ago. Note that in passing the year to Julian, nothing is assumed.
That is, yr = 89 DOES NOT equate to 1989 but 89 AD.

See JulianCvt for examples.

Name: JulianCvt Type: FUNCTION
Syntax: errc = JulianCvt(Ser&, mo, day, yr)

Reconstitutes a long integer formulated by Julian (qv) into a valid date.
The long integer must be a _TRUE_ julian date not an ordinal or
serialized date from an arbitrary point. The use of Date, Julian,
JualianCvt and/or DFrmat allow for extensive date caluclations. The
function return is non zero for unsupported dates (such as any BC date).
Example:

```
REM 1.   Find the maturity date for a 90 Certificate of
      '  Deposit purchased 4/5/1989

MatDate& = Julian(4, 5, 1989)     ' get julian date for 4/5/1989
CALL JulianCvt(MatDate& + 90, m, d, y)  ' convert it, add 90

PRINT USING " CD matures in 90 days on ##_/##_/#### ";m;d;y
CALL DFrmat(m, d, y, Mat$)
PRINT "That day is ";Mat$
```

Copyright (C) InfoSoft, 1991, 1992


```
REM  2. Calculate difference in 2 dates

CALL Date(tm, td, ty)           ' get today's date
Today& = Julian(td, tm, ty)      ' julian date for today

DueDate& = Julian(dd, dm, dy)    ' julian date for a past date

Diff& = ABS(Today& - DueDate&)   ' get difference


IF DueDate& > Today& THEN
   Text1.Text = "Library book is not due for "; Diff& ; " more days."
ELSE
   Text1.Text = "Library Book is "+ Diff& + " days overdue!"
   OverDue.Text = "Pay up $"+ (Diff& * LateChg)
END IF
```

Name: KBCapsOn / KBCapsOff Type: SUB
Syntax: CALL CapsOn
Syntax: CALL CapsOff

Neither of these take an argument or pass a parameter. They simply engage (CapsOn) or disengage (CapsOff) the Caps Lock state via the WIN API.

Name: KBInsOff / KBInsOn Type: SUB
Syntax: CALL KBInsOff
Syntax: CALL KBInsOn

This subroutine simply puts the keyboard into INSERT ON state (KBInsOn) or turns the insert toggle off (KBInsOff). Example:

CALL KBInsOn

Name: KBNumsOn / KBNumsOff Type: SUB(s)
Syntax: CALL KBNumsOn
Syntax: CALL KBNumsOff

Sets the Keyboard Num Lock key to on (NumsON) or off (NumsOff).

Name: KBScrLkOn / KBScrLkOff Type: SUB
Syntax: CALL KBScrLkOn
Syntax: CALL KBScrLkOff

Toggles the scroll lock key on or off. If the keyboard is equipped with LED's, they are also toggled to the appropriate state.

Copyright (C) InfoSoft, 1991, 1992

Name: KeyLock? Type: FUNCTION
Syntax: status = KeyLockC()
Syntax: status = KeyLockN()
Syntax: status = KeyLockS()
Syntax: status = KeyLockI()

This queries the WIN API to see which keyboard locks are engaged. One

routine is provided each for Caps, Nums, Scroll and Insert keys and all return zero for NOT engaged, non zero for engaged:

KeyLockC - Check Caps Lock KeyLockS - Check Scroll Lock
KeyLockN - Check Num Lock KeyLockI - Check Insert key

Example:

```
IF KeyLockC THEN Text1.text= "Caps Lock is ON!"
```

Name: LCount Type: FUNCTION
Syntax: NumLines = LCount(ByVal Fil$)

This is a handy routine that quickly scans an existing disk text file for ASCII 13 (carriage return) and counts them. This is handy for sequential file I/O operations where you might want to inform the user how long processing will take or to figure the size an array needs to be to hold the file contents.

LCOUNT is a function that requires only a valid file name. Unlike the QB4.5 version of GLib, you need not specify a scratch buffer - WGLib creates a temporary internal buffer.

Note: LCount starts counting CR's at the current file position so you can count only a portion of the file by using SEEK or FPtrSet to set the file pointer to other than the start of the file. After the operation, LCount closes the file. If you need access to it, simply re open it with FOPen or VB's OPEN statement. LCount is incredibly fast: a 50K test file takes only .5 seconds to count on a 286 system.

Example:

```
DECLARE FUNCTION LCount%(ByVal fil$)
..
..
fil$="GLIBDEMO.BAS"             ' text, not QB quick save format!

LineCount = LCount(fil$)
IF LineCount > 0 THEN
   GOSUB CalcTime
   Text1.text= "File will take "; CalcResult ;" secs to process."
ELSE
   GOSUB ErrControl
END IF
```

Copyright (C) InfoSoft, 1991, 1992

Name: LastNFirst Type: FUNCTION
Syntax: Swapped$ = LNameF(ByVal text$)

This data entry routine is handy for rearranging names from some other source to convert them to "Lastname, FirstName". It works on names of all types, those with middle initials, just first and middle initials and multiple middle names. It does not work correctly on Jrs, people who are the II, III or IV (ad nauseum). In this case, I'd suggest using BASIC's INSTR and LEFT$ to trim off the Jr or II etc and append it after the LNAMEF call.

All that is required is that the name have no leading or trailing spaces. You can trim these spaces without either creating a new string variable OR altering the original as shown in the example. The return is the names in swapped format. Examples:

"Mary Beth J. Sandra Brooks" => "Brooks, Mary Beth J. Sandra"
"P. T. Barnum Bailey" => "Bailey, P. T. Barnum"
"Thomas Q. McFly III" => "III, Thomas Q. McFly"
"John Public" => "Public, John"

```
Example:
DECLARE FUNCTION LNameF$(ByVal text$)
..
..
text$="John Smith   "       ' MUST NOT have trailing space...

' Look for Jr or a sequel indicator at the end
IF INSTR(LEN(text$) - 5,text$, "IVJrR") THEN
    GOSUB ReformatName              ' hack it off, if so
END IF

NewName$ = LNameF$(LTRIM$(RTRIM$(text$)))   ' passed trimmed copy
PRINT text$                         ' "Smith, John"
```

Name: MHZ Type: FUNCTION
Syntax: speed = MHZ()

This returns an approximation of the speed the system is operating at. This return is the result of a very quick benchmark. It is an "approximate" Mhz because it will be off a little depending on the number of wait states of the machine, and some faster PC/8088 clones return falsely high numbers.

MHZ returns the speed factor as a whole number: that is a return of 935 means an effective MHZ speed of 9.35. Divide the return by 100. It is advised that you cross reference the speed with the chip: if MHZ returns 1400 but CPUINFO indicates a 8088 machine, you know the speed is wrong and that the PC is more likely 8 to 10 MHz.

Copyright (C) InfoSoft, 1991, 1992

Name: MaxI, MinI Type: FUNCTION
Syntax: ret = MaxI(ByVal a%, ByVal b%)
ret = MinI(ByVal a%, ByVal b%)

Returns the larger (MaxI) or smaller (MinI) of 2 passed Integers. See also MaxIArry/MinIArry.

Name: MaxL, MinL Type: FUNCTION
Syntax: ret = MaxL(ByVal a%, ByVal b%)
ret = MinL(ByVal a%, ByVal b%)

Returns the larger (MaxL) or smaller (MinL) of 2 passed Long Integers. See also MaxLArry/MinLArry.

Name: MaxIArry, MinIArry Type: FUNCTION
Syntax: ret = MaxIArry(Arry%(), ByVal NumEls)
ret = MinIArry(Arry%(), ByVal NumEls)

Scans an entire integer array from starting with the element passed to find and return the largest (MaxIArry) integer foundor the smallest (MinIArry). The second parameter indicates the number of elements to search. Example:

```
REDIM ProgDat%(1000)
..
..
' scan for largest int in the elements 250 - 750 elements
smallest = MinIArry(ProgDat(250), 750)
```

Name: MaxLArry, MinLArry Type: FUNCTION
Syntax: ret = MaxLArry(Arry%(), ByVal NumEls)
ret = MinLArry(Arry%(), ByVal NumEls)

Scans an entire Long array from starting with the element passed to find and return the largest (MaxLArry) integer foundor the smallest (MinLArry). The second parameter indicates the number of elements to search. Example - see MinIArry, MaxIArry.

Name: MemCompA Type: FUNCTION
Syntax: offset = MemCompA(Src(beg), Dst(beg), ByVal words%)

Very quickly compares 2 arrays (or two sections of the same array) and returns the first offset where the 2 arrays do not match. You will find that the declaration file WGLIB.BAS contains 2 declarations for this

routine. The first allows for fast integer array compares, the second allows for the same compares on any array (even user defined, TYPE structures). Note that the WORDS parameter designates how many items to compare by indicating the number of words (16 bit integers) to compare. Be sure to use the proper multiple for Currency, doubles etc!

Copyright (C) InfoSoft, 1991, 1992

Name: NybbleCombine Type: FUNCTION
Syntax: Byte = NybbleCombine(ByVal HiNybble, ByVal LoNybble)

This combines 2 values representing the high and low nybbles into a single byte (half an integer). NybbleSplitLo and NybbleSplitHi perform the complementary operation by splitting the value of a byte into separate low and high nybbles.

Name: NybbleSplitLo / Hi Tyep: FUNCTION
Syntax: Nibble = NybbleSplitLo(ByVal byte%)
Nibble = NybbleSplitHi(ByVal byte%)

These nybble split operations return either the low (NybbleSplitLo) or high (NybbleSplitHi) portions of a byte. When used with NybbleCombine, can be used to combine record info in database operations.

Name: ParseFileSpec Type: FUNCTION
Syntax: errc = ParseFileSpec(ByVal raw$, FileInfo As Any)

Given a legal filename, this will parse it into a definite TYPE structure of it's component parts. The format of the structure is as follows:

```
TYPE struct
   Drv AS STRING * 2
   Path AS STRING * 64
   Fil AS STRING * 8
   Ext AS STRING * 3
END TYPE
DIM FInfo AS struct
```

Of course, while you can use any names you wish, the sizes must remain as shown. The dot separating Fil and Ext is omitted, but path backslashes and the drive colon are preserved. If ParseFileSpec finds what it perceives to be an invalid character or size in the passed raw file string, it will do what it can and return an error code of -1.

Name: PCase Type: Function
Syntax: Prop$ = PCase$(ByVal p$)

Converts a passed string to 'proper case'. That is, "bob smith" is returned as "Bob Smith". Prior to calling PCASE convert the string to lower case and check that the string to convert is not a null string:

```
x$ = "TIMOTHY FOOBAR"
x$ = LCASE$(x$)
IF LEN(x$) THEN p$ = PCase$(x$)     ' or p$ = PCase$(LCASE$(x$))
```

Copyright (C) InfoSoft, 1991, 1992

Name: ReverseStr Type: Function
Syntax: BackWard$ = ReverseStr(ByVal s$)

Reverses the character sequence in a string very quickly. When used in conjunction with XLATE, that can be a failry good encryption system.

Example:

```
x$ = "PassWord"
PW$ = RevStr(x$)            ' returns as "droWssaP"
```

Name: RINSTR Type: FUNCTION
Syntax: position = RINSTR(ByVal test$, ByVal ch$)

Returns the LAST occurrence of a character in a string. This works conversely to BASIC's INSTR, which returns the first occurrence, and is faster and much more code efficient than a loop to keep testing INSTR until the end of the string is reached. The character location however is still returned from the left. Note: RINSTR works only on character seeks, not on sub strings like INSTR does. That is, in the case of

```
j = RINSTR("ABCDEFG","A@B")
```

RINSTR will only seek and match on "A", not "A@B". Multiple passes thru RINSTR, however seeking each successive character in a sub string could be accomplished. Example:

```
DECLARE FUNCTION RINSTR%(searched$, seek$)
..
..
test$="123456x890" : char$="x"
l = RINSTR(test$, char$)                ' returns 7
```

Name: ShiftLeftI/ShiftRightI Type: FUNCTION
Syntax: Result = ShiftLeftI(ByVal value, ByVal ShiftCount)

Result = ShiftRightI(ByVal value, ByVal ShiftCount)

This returns RESULT after the integer VALUE has been shifted left or right the number of times indicated in ShiftCount. Besides providing a quick multiplication and division method, it is handy for bit operations such as isolating the date bits in a DATE word for instance.
See also ShiftRightL (etc) for shifting LONG integers.

Name: ShiftLeftL/ShiftRightL Type: FUNCTION
Syntax: Result& = ShiftLeftL(ByVal value&, ByVal ShiftCount)
Result& = ShiftRightL(ByVal value&, ByVal ShiftCount)

This works identical to ShiftRightI and ShiftleftI except both RESULT and VALUE are long integers.

Copyright (C) InfoSoft, 1991, 1992

Name: StrCmpI Type: FUNCTION
Syntax: result = StrCmpI(ByVal str1$, ByVal str2$)

Retuns an integer code indicating if the strings are identical after performing a case insensitive compare. Results return:
< 0: string 1 is 'less'
= 0: strings are equivalent
> 0: string 1 is 'greater'

Name: SubDirCount Type: FUNCTION
Syntax: count = SubDirCount(ByVal mask$)

Count the number of subdirectories matching a given mask. This counts the number of files with the directory attribute in the default directory. Note that while it is unusual, a directory CAN have an extension. Example:

```
DirCnt = SubDirCount("*.*")
```

Name: SubDirCH Type: FUNCTION
Syntax: errc = SubDirCH(ByVal SubDirName$)

Name: SubDirMK Type: FUNCTION
Syntax: errc = SubDirMK(ByVal SubDirName$)

Name: SubDirRM Type: FUNCTION

Syntax: errc = SubDirRM(ByVal SubDirName$)

Changes (SubDirCH), Makes (SubDirMK) or Removes (SubDirRM) the sub directory specified by SubDirName$. The advantage over BASIC's native sub dir functions is case of an error, rather than an error CONDITION being forced on your code, an error CODE is returned. The return code is set (non zero) if an error is encountered.

Name: SubDirExist Type: FUNCTION
Syntax: RetCode = SubDirExist(ByVal mask$)

Returns a non zero value if a given sub directory exists, and zero if it does not.
Example:

```
IF SubDirExists("QB45") THEN
   CHDIR "QB45"
ELSE
   PRINT "No can do"
END IF
```

Copyright (C) InfoSoft, 1991, 1992

Name: Swap? Type: SUBS
Syntax: CALL SwapI(a%, b%)
CALL SwapL(a&, b&)
CALL SwapS(a&, b&)
CALL SwapD(a&, b&)
CALL SwapC(a&, b&)

Visual Basic lacks an instrinsic function to swap or swicth variable contents. WGLib provides these in 5 versions to swap variables in any of the 4 data types integer (SwapI), long integer (SwapL), single precision (SwapS), double (SwapD) or currency (SwapC). Any return from these should be ignored as meaningless.

Name: SwapStr Type: SUB
Syntax: CALL SwapStr(a$, b$)

Performs a compleat and comprehensive swap of 2 strings.

Name: SysTicks Type: FUNCTION
Syntax: TicksSoFar& = SysTicks&()

SysTicks returns the number of clock ticks that have elapsed since midnight. This allows for an even finer resolution of time that SysTime or TIMER. Note that SysTicks returns a LONG integer.

Name: TimerToggle Type: SUB
Syntax: CALL TimerToggle(ByVal TimerNum, ByVal Toggle)

TimerToggle and TimerElapsed provide for an assembler based medium resolution set of timers. The granularity of these timers is approximately smaller than BASIC's TIMER function which is seconds based.

The TimerToggle and TimerElapsed functions allow for 5 different time slots so that you can track several different processes. When invoking TimerToggle, TimerNum should be 1 to 5 indicating the timer to toggle.

The Toggle parameter indicates whether you are starting or stopping that timer: 0 HALTS or stops the timer, any other value starts it. If a given timer has already been started, issuing another START command for it causes the previous start time to be overwritten - a timer cannot be RE-started.

Copyright (C) InfoSoft, 1991, 1992

Name: TimerElapsed& Type: FUNCTION
Syntax: ProcTime& = TimerElapsed&(ByVal TimerNum)

Fetches the number of elapsed clock ticks since the specified timer was started. If the timer was never started, garbage is returned. TimerNum refers to an integer 1 to 5 indicating which timer elapsed time is to be returned. TimerElapsed& returns a LONG INTEGER, so be sure to use the right data type. Also TimerElapsed& does not do an implicit timer stop function, it merely returns the difference of the start and stop ticks for that timer. Example:

```
CALL TimerToggle(1,1)          ' start timer one
..
CALL TimerToggle(1,0)          ' stop timer one
ProcTime& = TimerElapsed&(1)    ' get elapsed timer ticks
                '   into ProcTime&
```

Name: VBLoaded Type: FUNCTION
Syntax: Result = VBLoaded()

This simply checks to see if the currently executing program is VB.EXE. The use for this is esoteric but allows you to determine of the process executing in memory is an .EXE or a program image being executed by VB.EXE. Possible returns are:

0 VB Not loaded - EXE file executing
1 VB.EXE loaded

Name: ValidDrv Type: FUNCTION
Syntax: result = ValidDrv(ByVal drv$)

ValidDrv tests a given character passed to see if it is possibly a valid drive character. The return is 0 or non zero indicating if the drive is valid and available. The spectacular thing about this function is that it returns LOGICAL, not just physical drives, so that if the DOS version is greater than 3.0, if SUBST or networking software are in use, ValidDrv will return correct information - this is done by accessing the IOCTL functions if DOS 3.0 or greater is active. The only possible non true return would be drive B: in which, the system recognizes A: as B: when only one floppy is installed.
Example:

```
DECLARE FUNCTION ValidDrv%(ByVal a$)
..
..
FOR x = 1 TO 26
   msg$ = CHR$(x+64);
   IF ValidDrv(CHR$(x+64)) THEN
      msg$ = msg$ + " is a valid, active drive letter."
   ELSE
      msg$ = msg$ + " is not a valid drive letter."
   END IF
NEXT x
```

Copyright (C) InfoSoft, 1991, 1992

Name: VARPTR Type: FUNCTION
Syntax: vptr = VARPTR(thing)

One of the things left out in VB from other Microsoft BASIC implementations was VARPTR. This returns the offset of a variable or structure. Useful in some converting some older QB code to VB, but be careful! Winows managemes memory somewhat differently than DOS, which

may be impacted by direct memory acces.

Name: VARSEG Type: FUNCTION
Syntax: vseg = VARSEG(thing)

Returns the memory segment of a passed variable or structure. This is a mainstay of vaious DOS QB versions but was not implemented in VB.

Name: VerifyGet / VerifySet Type: FUNCTION / SUB
Syntax: vflag = VerifyGet()
Syntax: CALL VerifySet(ByVal vflag)

Sets or gets the DOS VERIFY switch. This is not a 'read after write' operation as is sometimes thought, but DOS will check the CRC of the data written and compare it to the source. Note that if you intend to alter such a system switch, it is good programming practice to restore it to its original setting when your program terminates, this switch can be determined via VerifyGet. In calling VerifySet, 0 turns VERIFY OFF, 1 turns it ON. Example:

```
Vflag = VerifyGet
..
..
CALL VerifySet(VFlag)
```

Name: ValFileName Type: FUNCTION
Syntax: errc = ValFileName(ByVal fil$, DOSCode)

This checks a string you pass it to determine if the string is a valid filename. Pre testing a string that you may have gotten from end user input, helps avoid runtime errors later on and in the case of novice end users allows, considerable feedback from your program on what is wrong with a filename typed in.

The process is twofold - it tests for characters such as ,[>< and also attempts to open the file and returns 2 error codes. In the case of the character test, the FUNCTION returns 0 if it finds no offending characters, or the ASCII value of any invalid filename character. This allows you to be able to give apparently intuitive feedback to users on valid filename characters.

Copyright (C) InfoSoft, 1991, 1992

A second pass is needed to test drive and path validity. To test this, ValFileName attempts to open or create the file and any DOS error will be

in the parameter DOSCode:
   3 - Drive or path not found
   4 - No handle available ("Too Many Files")
   5 - Access denied (already opened on multi system)
   80 - file exists.

   NOTES:
   1) VFNAME does NOT actually create or open the file! It just pre-tests for any possible runtime error in trying to do so.
   2) VFNAME requires DOS 3.x
Example:

```
   DECLARE FUNCTION VFName%(ByVal fil$, DOScode%)
   ..
   getfname:
     fil$ = Text1.Text = fil$

     CharCode = VFName(fil$, DOSCode)
     IF CharCode THEN
        ret = MsgBox("Sorry, but '"+CHR$(CharCode)+"' is illegal in
                  filenames!",48,"Ooops!")
     END IF
```

      Name: VLabelGet   Type: FUNCTION
   Syntax: VLabel$ = VLabelGet$(Drive)

Allows you to get the volume label on any attached disk: (0=default drive, 1 = A:, 2 = B: etc). Volume Labels are always 11 characters, so when using VLabelGet you will get back an 11 character string regardless of the number of characters in the string.

      Name: WinCurTime   Type: FUNCTION
   Syntax: ticks& = WinCurTime

Retrieves the current WINDOWS time.

      Name: WinDirectory$ Type: FUNCTION
   Syntax: win$ = WinDirectory$()

Returns a string representing the directory where WINDOWS resides.

Copyright (C) InfoSoft, 1991, 1992

Name: WinKBFuncKeys
Syntax: NumFKeys = WinKBFuncKeys()

Returns the number of function keys on the installed keyboard. This can be handy in reassigning F11 and F12 if they are not on the KB.
Return: 1 = 10 Function keys 4 = 12
2 = 12 (sometimes 18) 5 = 10
3 = 10 6 = 24

Note: This is as per the WINDOWs documentation, but at least WIN 3.1 seems to return the actual number of function keys (10, 12, 18 etc).

Name: WinKeyBdType type: FUNCTION
Syntax: kbtype = WinKeyBDType()

Returns an integer code indicating the type of keyboard driver intalled in WINDOWS. 1 = IBM PC/XT 83 key style
2 = Olivetti 102 keyboard
3 = IBM/AT 84 compatible
4 = IBM enhanced 101/102 compatible
5 = Noika 1050
6 = Noika 9140

Name: WinMem Type: FUNCTION
Syntax: mem = WinMem&()

Returns the amount of system resources available (free). This is a long integer and includes RAM as well as swap files memory.

Name: WinMode Type: FUNCTION
Syntax: WMode = WinMode()

Returns the mode that WINDOWS is operating in, Standard or Enhanced. Since VB will not run in REAL mode, this mode is not queried nor returned. A return of 1 = Standard mode while 2 = Enhanced.

Name: WinMouse Type: FUNCTION
Syntax: ret = WinMouse()

Simply polls the WIN API to see if a mouse is installed and/or operable. The return is zero for false, non zero for true.

Copyright (C) InfoSoft, 1991, 1992

Name: WinPrgName Type: FUNCTION
Syntax: EXEName$ = GetWPrgName$

WinPrgName polls the WIN API and returns a string with the drive/pathname of the currently executing program. (See also ArgCnt and ArgVar). Note: WinPrgName will return different information in the VB environment than as a .EXE file, because the program running is VB.EXE and NOT your program. Example:

```
EXEName$ = GetWPrgName$
' returns "C:\DIRNAME\FOOBAR.EXE"
Text1.text = "Program running is: "+ EXEName$
```

Name: WinSysDir$ Type: FUNCTION
Syntax: SystemDir$ = WinSysDir$()

Returns a string indicating the pathname of the Windows SYSTEM directory, typically "\WINDOWS\SYSTEM".

Name: WinSpkrSnd Type: FUNCTION
Syntax: WinSpkrSnd(ByVal freq%, ByVal dur%)

Sounds the speaker for the specified duration in the specified frequency. Note that while this has the same syntax as SpkrSnd in our GLib for QB/QBX, it is radically different in that it performs it's speaker operations via the WIN API.

Name: WinTempDirectory$ Type: FUNCTION
WinTempDrive$
Syntax: TempDrv$ = WinTempDrive$()
TempDir$ = WinTempDirectory$()

Returns a string representing the optimum drive and or directory for creating temporary files.

Name: WinVer Type: FUNCTION
Syntax: WVer = WinVer()

Returns the version of WINDOWS running. The return is a whole number and should be divided by 100 to get the true minor version.

Copyright (C) InfoSoft, 1991, 1992

Name: XLate Type: FUNCTION
Syntax: errc = XLate(source$, ByVal table$)

This will translate or substitute all characters in SOURCE$ from the list of characters in TABLE$ based on their ASCII value. Note that since this may be as high as 255, that TABLE$ should allow for all possibilities and be 256 characters long. This provides for an easy and configurable encryption scheme.

```
Example:
FOR x = 1 to 256
   Table$ = Table$ + CHR$(256 - x)
NEXT x
Serial$ = "123456"              '  '1' becomes ASCII 206
errc = XLate(Serial$, Table$)   ' '1' = 49 and 255 - 49 = 206.
```

Name: ZellerDay Type: FUNCTION
Syntax: DCode = ZellerWeek(ByVal month%, ByVal day%, ByVal yr%)

This uses Zeller's Congruence to determine the day of week for any valid date. Example:

```
WeekDay = ZellerDay(5, 17, 1991)      ' returns 3 for Wed

Text1.Text = DayName$(WeekDay)
```

Copyright (C) InfoSoft, 1991, 1992

II. VB Custom Properties Actions

The following are specially designed routines that allow you to perform special operations on many of the controls that come with Visual BASIC. Note that while these are not actually properties, if carefully called, they can appear to be such.

Name: CBClearList
Syntax: CALL CBCLearList(Ctl AS CONTROL)

This allows you to instantly clear the contents of a Combo Box. Rather than looping thru each element to set the string to NULL this calls directly upon the the WIN API to reset the contents of the control passed thereby freeing that memory with a minimum of hassle.

Name: CBShowList
Syntax: CALL CBShowList(Combo AS CONTROL)

Forces the list attached to a Combo Box to drop down, thereby preventing the end user from having to click on the open button to see the choices. For maximum visual effect, this should be called after the size of the combo list has been established. An ideal location for this is in the GotFocus Event.

Name: FrmFlash
Syntax: CALL FrmFlash(ByVal hWnd)

Flashes a form or window as if it has the focus, then turns it off. The flash is that of the borders and title bars assuming their 'active' color. This can be especially effective in background type DDE message windows to indicate the contents have changed. Note that the parameter required, hWnd or a handle to a window or form, is not available from VB, but is via our GetCtrlHnd. Also, hWnd is a reserved word in VB.

Name: LBClearList
Syntax: CALL LBCLearList(Ctl AS CONTROL)

Like CBClearList, this allows you to instantly clear the contents of a List Box. Rather than looping thru each element to set the string to NULL this calls directly upon the the WIN API to reset the contents of the control passed thereby freeing that memory with a minimum of hassle.

Copyright (C) InfoSoft, 1991, 1992

Name: GetCtrlHWnd%
Syntax: WHandle = (AnyVBControl AS Control)

There are many things you can do by calling the WIN API directly from Visual BASIC, but most, if not all, of them require that you pass the hWnd (WINDOWS handle) as part of the syntax, yet VB provides no easy access to hWnd. The Window (or control) Handle can be gotten in VB but it is very slow and cumbersome. GetCtlHWnd provides easy access tot eh control handle. Note: This is a very advanced routine and should only be used by those who have a documented source of WIN API calls.

Name: LBCLearList
Syntax: ignore = LBClearList(ListCtl As Control)

Once a list box has been created, it is cumbersome and slow to delete all the items in it solely using VB. LBClearList allows you to instantly clear the contents of a list box with one line of code.

Name: LBFindPreFix

Syntax: index = LBFindPreFix(Ctl As Control, ByVal Search$)

Passed a string, this will search the contents of a list box and return the index of the first element that begins with the string passed.

Note that this is not an INSTR type search, but it finds the element that starts with Search$. Use of this WIN API function is how WinHelp finds and updates the current selection as you type in characters in the SEARCH box. The function returns the index os the element with matches the passed string.

Name: LBGetFirst
Syntax: index = LBGetFirst(ListCtl As Control)

Passed a valid list box control, this funtion returns the index of the first visible item. See also LBSetFirst.

Name: LBSetFirst
Syntax: errc = LBSetFirst(ListCtl As Control, ByVal Index)

Passed a valid ListBox control, this sets the first visible element to that of INDEX.

Copyright (C) InfoSoft, 1991, 1992

Name: TextBoxFLoad
Syntax: retc = TextBoxFLoad(TextCtl As Control, ByVal fil$)

One of ther most excruciatingly slow aspects of VB and Control manipulation is loading a file, one line at a time into a (MultiLine) ListBox. TextBoxFLoad cuts out one of the 'middlemen' by bypassing most of the VB API and goes directly to the WIN API to load the specified file directly into the TextBox. By avoiding VB and being in assembler, the time saved in the loadin process is incredible.

Name: TBSizeLimit
Syntax: retc = TBSizeLimit(TextCtl As Control, ByVal Size%)

While text boxes are neat as ready made input controls, they lack a great

deal of functionality. One thing missing is the ability to limit the number of characters input. TBSizeLimit allows you to specify in SIZE the maximum number of characters to accept. For those familiar with GLIB for DOS style QBs, this is like MFed's FSize parameter. See also TBPassword.

Name: TBPassWord
Syntax: TBPassWord(TextCtl As Control, ByVal Char%)

The invocation of this function allows you to specify the echo to echo when the end user types in a character, allowing you to devise a password style input box. Char is the ASCII code of the character to echo.

Copyright (C) InfoSoft, 1991, 1992

III. Visual Basic Custom Controls

1. Clock

ä▫▫▫▫▫▫▫y▫Ð▫¤▫▫▫q▫▫▫▫▫▫▫>…DEL*▫▫(▫▫▫ÿ è▫è▫▫▫▫▫▫▫▫▫▫▫▫▫▫PBrush▫▫▫▫▫▫
▫▫▫0 0 48
28▫@▫▫▫BM▫▫▫▫▫▫▫▫▫v▫▫▫(▫▫▫0▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫ ▫▫▫Ä▫▫▫Ä▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫€▫▫€▫▫▫€
€▫€▫▫▫€▫€▫€€▫▫€€

€ÀÀÀÿÿÿ
ÿðÿÿÿÿÿÿÿÿÿðˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆ€ÿÿÿˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆ
ÿÿðˆ‰™™˜ˆˆˆ‰™™™˜ˆ™™™˜ˆ€ÿðˆ™™™™˜ˆˆ™™™™™˜‰™™™™ˆ€ÿðˆ‰™™™˜ˆˆ™ˆˆ
‰˜‰˜ˆˆ™ˆ€ÿðˆˆˆˆ‰˜ˆˆ™ˆˆ‰˜‰˜ˆˆ™ˆ€ÿðˆˆˆˆ‰˜ˆˆ™ˆˆ‰˜‰˜ˆˆ™ˆ€ÿðˆˆˆˆ‰˜™ˆ™ˆˆ‰˜
‰˜ˆˆ™ˆ€ÿðˆˆˆˆ‰˜™ˆ™ˆˆ‰˜‰˜ˆˆ™ˆ€ÿðˆˆ‰™™˜™ˆ™ˆˆ‰˜‰˜ˆˆ™ˆ€ÿðˆˆ‰™™˜ˆˆ™ˆˆ‰˜
‰˜ˆˆ™ˆ€ÿðˆˆˆˆ‰˜ˆˆ™ˆˆ‰˜‰˜ˆˆ™ˆ€ÿðˆˆˆˆ‰˜ˆˆ™ˆˆ‰˜‰˜ˆˆ™ˆ€ÿðˆˆˆˆ‰˜™ˆ™ˆˆ‰˜
‰˜ˆˆ™ˆ€ÿðˆˆˆˆ‰˜™ˆ™ˆˆ‰˜‰˜ˆˆ™ˆ€ÿðˆˆˆˆ‰˜™ˆ™ˆˆ‰˜‰˜ˆˆ™ˆ€ÿðˆ‰™™™˜ˆˆ™ˆˆ‰˜
‰˜ˆˆ™ˆ€ÿðˆ™™™™˜ˆˆ™™™™™˜‰™™™™ˆ€ÿðˆ‰™™˜ˆˆˆ
‰™™™˜ˆ™™™˜ˆ€ÿÿˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆÿÿÿðˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆ€ÿÿÿðÿÿÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
ÿÿ˜ˆ

Clock is a Custom Control for Visual BASIC that allows you to constantly display a digital clock on any form or window. This approach to time display elimiates the need for you to develop a timer routine and update the screen.

While this is a fairly intuitive and straight forward control, it is necessary to note that WINDOWS allows for only 30 timers to be active at a time. While 30 is a lot, you must allow for this depending on the number of instances of the CLOCK custom control, the use of the WINDOWS on screen clock and the number of TIMERS you have in use.

Standard Properties supported:

CtlName	Width	DragIcon
Index	Height	Tag
BackColor	Visible	FontName
ForeColor	Enabled	FontSize
Left	Parent	
Top	DragMode	

Custom Properties supported:

AlarmEnable	TimeFormat
AlarmHr	TimeLabel
AlarmMin	TimerMode
HideSecs	TimeZone

Custom Events supported:

Alarm

Clock Custom Properties Defined

Name: Alarm_Enable Type: BOOLEAN

When enabled, and the system time matches that defined as the alarm time (Alarm_Hr: Alarm_Min), the Alarm Event is triggered. The event is triggered each second for one full minute - until the system time no longer matches that of the Hr:Min designated as the alarm time.

It is up to your program to supply the actual activity to be performed at the specified time. It may be as simple as a series of beeps (like in an appointment book application), or it may signal that it is time to exit your application and execute another (such as a timed EMail link).

Copyright (C) InfoSoft, 1991, 1992

Name: Alarm_Hr Type: INTEGER

Defines the hour that the alarm will sound. Must be used in conjunction with Alarm_Enable and most likely Alarm_Min.

Name: Alarm_Min Type: INTEGER

Defines the minute that the Alarm Event will trigger. This is only of any matter if Alarm_Enable is active (TRUE) and will likely be used in conjunction with Alarm_Hr.

Name: HideSecs Type: BOOLEAN

This TRUE/FALSE property allows you to hide the display of seconds in your clock display. In some instances it may be more aesthetically pleasing to simply display in Hrs:Min format rather than Hrs:Min:Sec.

There are several benefits in hiding the seconds display. First, the WIN API has to process many, many requests for action (called messages) from every program active or running and still contend with things that the end user may do such as rearrange windows on screen requiring any number of screen updates. To contend with the virtual blizzard of messages, the WIN API uses a queque type approach.

So, when you've installed the Clock Custom Control, while we've designed to to update the display more frequently than once per second the speed with which other processes act on their messages has an impact on how often Clock will actuall get it's messages. The effect of this on slower systems, such as 80286 systems means that the time could 'jump' more than one second occasionally.

Next, a future version of Clock will implement a less demanding timer when HideSecs is TRUE, thereby freeing up system resources for other processes running.

Name: TimerMode Type: BOOLEAN

Rather than displaying the current time with the default TimerMode as FALSE, you can tailor Clock to display the time elapsed (TRUE) from a desired starting time. In this mode, the TimeFormat setting is ignored and the time display will ne in hh:mm format, but the HideSecs setting

will be honored. Naturally, the Alarm Property is not available.

The Enabled standard properties have a special impact on the ElapsedTimer property. When FALSE and Clock is not in TimerMode, the currect time is simply displayed. However, more extensize control is required for Elapsed Mode, specifically what the starting time is. By default, the starting time will be the time that the Clock/Timer was created, however to supress the timer display until you are ready, you

Copyright (C) InfoSoft, 1991, 1992

can either set the VISIBLE property to FALSE or set the ForeGround Color to that of the BackGround. Reversing either of these will normalize the display. Then to set a new starting base time, simply set the ENABLE property to 1. Each time your code sets or resets the ENABLED property to 1, the starting time is reset.

Secondly, if the Clock is in Elapsed Timer Mode, but is not Enabled, it may be desirable to hide the display by setting the Standard Visible property to FALSE.

Name: TimeFormat Type: ENUMERATED

At design time, you can designate the format of the Clock display. The choices allow you to display the hours in either 12 or 24 hour format.

Name: TimeLabel Type: ENUMERATED

Allows you to designate the Clock's time display label. You may choose from AM/PM, am/pm or None.

Name: TimeZone Type: INTEGER

The TimeZone property allows for each Clock display to represent the local time in another time zone. Simply initialize the value to the time difference for the desired locality. To display Eastern Time when the system is in the Central time zone, set TimeZone to -1; to display Mountain Time from the same zone TimeZone would be set to 1 and naturally. To display the time of most MidEast countries, subtract 1200 years.

In all instances, local time would be displyed by setting TimeZone to 0. Further, the value of TimeZone should be in the range 0 to 24.

Name: Alarm Clock Custom Event

The Clock Custom Control will trigger the Alarm Event when the following are true:

- AlarmEnable is true
- AlarmMin equals the current system time minute
- AlarmHr equals the current system time hour

If any ONE is not true the Alarm will not sound. The Event will be triggered approximately once per second, or until your code invalidates one of the above conditions. (No, we don't have a Snooze property).

Copyright (C) InfoSoft, 1991, 1992

2. Meter

ä§Ìh¡*(ÐPBrush
0 0 53
31àBMÚv(5dÄÄ€€€
€€€€€€€
€ÀÀÀÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿöfÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
ÿÿÿÿÿýÝÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿû»ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿöfÿÿø^^^^^^^^^^^^^^^^^^^^^^ÿÿÿÿÿ
wwwwwwwwwwwwwwwwwwwwwww^ÿûÿÿÿxÿÿÿÿð
xÿÿÿÿðîîîîïÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿxÿÿÿÿðîîîîïÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿxÿÿÿÿðîîîîïÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿx
ÿýÝÿðîîîîïðÿÿÿÿðÿÿÿxÿû»ÿðîîîîïðÿðððÿÿÿÿxÿöfÿðîîîîïðÿÿÿÿÿÿÿðÿÿÿxÿÿÿÿð
îîîîïðÿÿÿÿÿÿðÿÿÿÿÿxÿÿÿÿðîîîîïðÿÿÿÿÿÿÿÿÿÿÿÿxÿÿÿÿðîîîîïÿÿÿðÿÿðÿÿÿÿxÿÿÿÿðîîîîïÿð
ÿÿÿÿÿÿÿxÿÿÿÿðîîîîïÿÿÿÿÿÿÿÿÿðÿÿÿxÿÿÿÿðîîîîïÿÿÿÿÿÿÿðÿÿÿÿxÿù™ÿðîîîîïðÿ
ÿÿðÿÿxÿò"ÿðîîîîïÿÿÿÿðÿÿÿÿÿxÿôDÿðîîîîïÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿxÿù™ÿðîîîîïÿÿÿÿÿÿÿÿÿÿÿ
ÿÿÿxÿó3ÿðîîîîïÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿðÿÿÿÿÿ
ÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

Meter is a Custom Control for Visual BASIC that allows you to display a meter of the percentage of a job that is done. The visual effect is that seen when installing WINDOWS where a horizontal bar slowly gets filled in showing the percentage of the job that is complete.

When defining the Fore and Back colors, it is the ForeGround color that fills in the bar as the percentage increases, replacing existing BackGround color.

When locating and sizing a Meter Control on your form at design time, it need not be a horizontal style bar, Meter supports vertical meters as well as an Inverse property. The Inverse property causes the Meter to act in reverse mode, that is it begins filled in (with the ForeGround color) and 'drains' to an empty bar.

Standard Properties supported:

CtlName	Width	
Index	Height	
BackColor	Visible	FontName
ForeColor	Enabled	FontSize
Left	Parent	
Top	Tag	

Custom Properties supported:
Units_Total
Units_Done
Inverse
VerBose

Custom Events supported:
None

Copyright (C) InfoSoft, 1991, 1992

Meter Custom Properties Defined

Name: Inverse Type: BOOLEAN

When TRUE, the Meter starts out filled with the ForeGround color and 'drains' to an empty Meter as the percentage complete increases. The default (FALSE) setting allows the Meter Control to fill with the ForeGround Color as the percentage increases.

Name: Verbose Type: BOOLEAN

When set to TRUE, as the percentage complete increases, the Meter Control is labeled with the actual percent. A FALSE setting leaves the

control unlabelled and percentage done (or percentage left) is simply implied from the display.

The mode of text display will 'clip' any text that will not fit in the Meter Control and can cause an ungainly appearance. This is more likely with narrowish vertical meters where '100%' may not fit, but is also likely with horizontal meters when either of the SpecialFX are used. See the SpecialFX Property for more information regarding this.

Name: UnitsDone TYPE: INTEGER

In order to display a meter of the percentage done, the internal control code must know the total number of parts in the job and what the current number of parts done are. These values are handled by the Custom Properties UnitsTotal and UnitsDone.

The Meter Control code will handle all the calculations and color the Meter Control the appropriate amount, you simply set the amounts of the 2 units--- properties.

The UnitsDone property is updated as your program completes portions of your process. When the WIN API notifies the Meter Control that UnitsDone has changed, the Meter Control code updates the meter display. Note that simply addressing UnitDone cause WIN API to send a message to the Meter Control, even if the new value is the same as the old. However, Meter checks to see if the percentage has changed and exits early if there is no change.

See UnitsTotal for more information.

Copyright (C) InfoSoft, 1991, 1992

Name: UnitsTotal Type: INTEGER

As explained in the description of the UnitsDone property, the Meter Control Code will calculate the percentage from the values of UnitsDone and UnitsTotal. What these represent however is up to you.

Being of the data type integer (short), valid values will range from 0 to 32,767. But the actual amounts can be representative and independant of what is actually being done. For example, setting UnitsTotal to 100, allows you to directly manipulate the percentage done since whatever you set UnitsDone to will be the percentage displayed in

the Meter Control.

In the case of an installation process, let's assume you have to copy 2,509,832 bytes from floppy disks and the number of bytes read so far is what you wish to base the UnitsTotal Property on. Since 2.5 million is beyond the legal range of UnitsTotal, simply divide by 100 and set the UnitsTotal Property to 25098. Then as each file is copied, simply divide the new aggregate copied amount by 100 and the Meter Control will be accurate. The chances of such a wide spectrum Meter actually registering 100 of over 35,000 units on a 6' or so Meter is remote. Given that files can vary widely in size a byte method is more accurate than a file count method.

Another example would be in loading an index or random access file. The UnitsTotal could be the size of the file in records, then every two, three or five records, depending on the size of UnitsTotal, you could update the UnitsDone property so as to inform the Meter Control that the display needs updating.

Copyright (C) InfoSoft, 1991, 1992

3. Extended Control Text Box (XtdEd)

ä▫▫▫▫▫▫▫§X▫¤▫▫▫ñ▫▫▫▫▫▫▫éš*▫▫(▫▫▫▫|▫

PBrush
0 0 40
28ÀBM¦v((0ÄÄ€€€
€€€€€€€
€ÀÀÀÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆˆÿÿÿwwwwwwwwwwwwwwwwwÿÿð
ÿÿðÿÿðÿÿÿÿÿÿÿÿÿÿÿÿÿÿðÿÿðÿÿÿÿÿÿÿÿÿÿÿÿÿðÿÿððÿÿÿÿÿðÿÿ
ððÿÿðððÿðÿÿðÿÿðððÿðÿÿðÿðÿðððÿðÿÿðÿÿÿðððÿðÿÿðÿÿ
ÿðððÿðÿÿðÿðÿÿÿðÿÿðÿÿðÿðððÿÿððÿÿðÿÿðÿÿðÿÿðÿÿðÿðÿÿððÿÿÿ
ÿÿÿðÿðÿÿðÿÿÿÿÿÿÿÿÿÿÿÿðÿÿðÿÿÿÿÿÿÿÿÿÿÿÿÿðÿÿðÿÿð
ÿÿ

Overview

By far, one of the popular and useful tools in our GLib toolbox for QB/QBX is that of MFED, a text input control routine. Virtually everything that can be controlled in collecting user input can be programmed in. While VB's TextBoxes go along way in replacing MFed, there are still some things that MFed can do that you cannot do with a text box. So, we set out to replicate the key MFed capabilities in a custom control Text Box, and added a few facets available under WINDOWS.

Several of the things that you can control in our XTDEdit control can be altered in a standard TextBox, but requires separate WIN API calls for each box - XTdEdit allows you to access them as built in properties.

Properties List

BACKCOLOR	FONTNAME	PARENT
FORECOLOR	FONTBOLD	TAG
LEFT	FONTITALIC	VISIBLE
TOP	FONTSTRIKE	TABINDEX
WIDTH	FONTUNDER	MULTILINE
HEIGHT	FONTSIZE	
TEXT	ENABLED	

(Custom)

ErrAlarm
MaxSize
InsertMode
BadCharList
NumsOnly
ReadOnly

Events Supported

CLICK (std)
CHANGING

Copyright (C) InfoSoft, 1991, 1992

Name: MaxSize (INTEGER)

Available at design and runtime, your code can control the length of the text string to be entered into an XtdEdit control. By setting the MaxSize property, the edit control rejects any characters beyond that length and optionally beeps based on the ErrAlarm property.

The MaxSize property can be set at design time via the VB drop down properties box. If one text box is doing multiple edits, your code can set the current maximum character length at run time.

An obvious use of this is in data entry applications where you wish to collect only a certain number of characters, such as for a filename.

Name: InsertMode Custom Property (BOOL)

One of the things that irks me about the standard WIN Text Box is that they all start in overwrite mode. Type one character and the previous entry is gone. Even for the experienced user, tapping a cursor key, then backing up to preserve the original text requires you to tap the Insert Key for a true inseret mode.

The InsertMode Custom property allows you to design the control to work as other TextBoxes do (FALSE) or to support InsertMode upon startup (TRUE).

When set to TRUE, the XtdEdit Control does not just emulate insert mode, but it sets the WIN API for it. That is, upon entry if the start up is Insert Mode TRUE, XtdEdit sends a message to the WIN API to toggle the Insert Key ON. For the balance of the time that the edit control has the focus, it monitors the Insert state to see if the user has toggled it off. Should the user do so, then conventional text box input behavior takes place. Otherwise the control remains in overwrite mode.

Name: NumsOnly Custom Property (BOOL)

In the case where the input your application requires should be numeric only, your only option with a standard text box is to examine the contents or .TEXT property after the control loses focus.

With XtdEdit's NumsOnly Property, when set to true, the control examines each character and accepts it only if it is numeric. This makes XtdEdit ideal for data base entry where any number of fields such as phone numbers, zip codes and the like are numeric only.

Copyright (C) InfoSoft, 1991, 1992

Name: BadCharList Custom Property (TEXT string)

Like with the entry of numeric characters for phone numbers, there are instances where there are characters that you just do not want in the entry. Filenames are one example, but with data that will be saved to a sequential file, you will likely not want any commas in it.

By listing or populating the BadCharList property with those characters that are forbidden in the entry, you instruct the XtdEdit control to reject these. If one is encountered, XtdEdit will discard it rather than add or append it to the text being entered.

Name: ReadOnly Custom Property (BOOL)

In some instances, it is desirable that the text in an edit box be Read Only. That is, it is for the user's information only and is not within their power to change it.

An ideal example of this would be where you might have a record being edited displayed in one XtdEdit box where the user is editing. Another however, may be another XtdEdit control where the original record is displayed for comparison. The latter control should ideally be a ReadOnly control so that even when it receives the focus the user cannot change the contents.

In this or other instances where you wish to display something to the user, without them changing it you can set the ReadOnly property to TRUE and even when the XtdEdit control receives the focus via the mouse or tab key sequence, they can scroll thru multi line controls and otherwise view the data, but cannot alter it.

Name: ErrAlarm Custom Property (BOOL)

This characteristic is one ported from MFed. Whenever a user violates the constraints that the above XTdEdit control properties impose on them, when ErrAlarm is set to TRUE, the speaker will sound.

Some typical situations:
- Entering a character found in BadCharList
- Attempt to edit or change a ReadOnly control
- Attempt to enter an Alpha character in a NumsOnly control
- Entering a character that will cause the length of the text to exceed MaxSize

Copyright (C) InfoSoft, 1991, 1992

CHANGING Custom Event

One of the slickest things about the Windows Help system is it's ability to dynamically update the Search list. Open the SEARCH window and as you type each character into the text box, the contents or subject list is updated. With a standard text box, replicating such behavior is difficult at best, but easy with XtdEdit.

After each characacter is qualified according to the custom properties above (NumsOnly, BadCharList etc), it is appended to the contents. At that instant (as much as anything happens instantaneously in Windows), a CHANGING event is triggered. This event therefore notifies your code that the contents of the edit control is being updated.

Replicating the behavior found in the WIN help system in your own application is then quite simple. In response to the CHANGING event, simply call LBFindPreFix to get the first item in a list box that matches the contents of the XtdEdit control (XtdEdit.Text). If need be you can also use LBSetFirst to force the List Box to display that item.

An ideal use and example of this would be in a Movie or VCR tape database. When searching for movies with a certain star or director, or even a title, as each character is entered you can respond in the CHANGING event to execute the code needed to display the applicable section of the list box.

4. Enhanced Scroll Bars (EhVScroll, EhHScroll)

ä□□□□□□□z¤□□Ð□□□1□□□□□□□û–*□□(□□□oè□è□□□□□□□□□□□□□□PBrush□□□□□
□□□0 0 79
48□□□□□BMö□□□□□□□v□□□(□□□O□□□0□□□□□□□□□□□□€□□□Ä□□□Ä□□□□□□□□□□□□□□□□□□□€□□€□□□€
€□€□□□€□€□€€□□€€
€□ÀÀÀ□□□ÿ□□ÿ□□□ÿÿ□ÿ□□□ÿ□ÿ□ÿÿ□□ÿÿÿ□ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ÷ÿÿÿÿÿÿÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□□□□□□ÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□wwww□ÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□ˆ€ˆ‡□ÿÿÿòÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□ˆ□□‡□ÿÿÿòÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿ□€□□‡□ÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□ˆ€ˆ‡□ÿÿÿóÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
ÿÿÿÿÿÿÿ□ˆ€ˆ‡□ÿÿÿóÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□ˆ€ˆ‡□ÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
ÿ□ÿÿÿ÷□ÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□□□□□□ÿÿÿóÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□ˆˆˆˆ
□ÿÿÿóÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□ˆˆˆˆ□ÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□ˆˆˆˆ□ÿÿÿ÷ÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□ˆˆˆˆ□ÿÿÿóÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□ˆˆˆˆ□ÿÿÿóÿÿÿÿÿÿÿÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□□□□□□ÿÿÿò□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ÿÿÿ□wwww□ÿÿÿó□wwwpˆˆˆˆ€÷
www□ˆˆˆˆˆˆ€÷www□ÿÿÿ□ˆˆˆ‡□ÿÿÿ÷□ˆˆˆpˆˆˆˆ€øˆˆ‡□ˆˆˆˆˆˆ€øˆˆ‡□ÿÿÿ□ˆˆˆ‡□ÿÿÿ÷□ˆ□ˆpˆˆˆˆ€øˆˆ‡□ˆˆˆˆˆˆ€øˆ□
‡□ÿÿÿ□ˆˆˆ‡□ÿÿÿ÷□€□ˆpˆˆˆˆ€øˆˆ‡□ˆˆˆˆˆˆ€øˆ□‡□ÿÿÿ□ˆˆˆ‡□ÿÿÿ÷□□□□pˆˆˆˆ€øˆˆ‡□ˆˆˆˆˆˆ€ð□□□□ÿÿÿ□ˆˆˆ‡□ÿÿ
ÿ÷□€□ˆpˆˆˆˆ€øˆˆ‡□ˆˆˆˆˆˆ€øˆ□‡□ÿÿÿ□ˆˆˆ‡□ÿÿÿ÷□ˆ□ˆpˆˆˆˆ€øˆˆ‡□ˆˆˆˆˆˆ€øˆ□‡□ÿÿÿ□ÿÿÿÿ□ÿÿÿ÷□ˆˆˆpˆˆˆˆ€øˆˆ
‡□ˆˆˆˆˆˆ€øˆˆ‡□ÿÿÿ□□□□□□ÿÿÿ÷□ÿÿÿðˆˆˆˆ€ÿÿÿÿ□ˆˆˆˆˆˆ€ÿÿÿÿ□ÿÿÿ□ˆˆˆˆ□ÿÿÿ÷□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□ÿÿÿ□ˆˆˆˆ□ÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ□ˆˆˆˆ□ÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

ÿˆˆˆˆÿÿÿöÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿˆˆˆˆÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿˆˆˆˆÿÿ
ÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿˆˆˆˆÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿˆˆˆˆÿÿÿ÷ÿÿÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿˆˆˆˆÿÿÿöÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿˆˆˆˆÿÿÿöÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿwwwwÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿˆ€ˆ‡ÿÿÿóÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿˆ€ˆ‡ÿÿÿóÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
ÿÿÿÿÿÿÿˆ€ˆ‡ÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ€‡ÿÿÿ÷ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
ÿÿˆ‡ÿÿÿöÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿˆ€ˆ‡ÿÿ
÷ÿÿÿ÷ÿÿÿ
ÿÿÿ÷

When we were developing the Changing event for the XtdEdit control, it occurred to us that the CHANGING event would be ideal for Scroll Bars.

Such an event, linked to Scroll Bars would make dynamic updates and fine tuning easier and faster for the user. So we developed a sub class of scroll bars that supported just that. We were somewhat dismayed that the VB Pro Toolkit turned out to have one as well, but since not everyone has $100 for a few controls, or needs the CDK or the Help Compiler, we decided to go ahead and release our Enhanced Scroll Bars with WGLib as well.

The WGLib toolbox supports vertical Enhanced Scroll Bars (EnVScroll) as well as horizontal ones (EnHScroll). Both work the same as standard scroll bars with the exception of generating a CHANGING event which you have the option of not reponding to.

Properties List

LEFT	ENABLED	MIN
TOP	PARENT	MAX
HEIGHT	TAG	VALUE
WIDTH	LARGECHANGE	
VISIBLE	SMALLCHANGE	

Events List

CHANGE
CHANGING

Properties

All the properties listed behave much the same as the standard properties for standard scroll bars do. You set the MIN and MAX range, SMALLCAHNGEs are generated by clicking the arrow, LARGECHANGEs are generated by clicking the scroll bar itself between the slider button (called the THUMB) and the arrow.

Copyright (C) InfoSoft, 1991, 1992

CHANGING Custom Event

The key enhancement of our Enhanced Scroll Bars is that each time the value changes, whether as a result of dragging the thumb, clicking the arrow or generating a LARGECHANGE by clicking on the Scroll Bar between the thumb and arrow, a CHANGING event is generated.

In responding to the CHANGING event, you can get the current Enhanced scroll bar setting via the En?Scroll.Value property and update whatever is being scrolled (lists, colors, values etc). This gives a much snappier appearance to the application since the focus no longer has to be lost by the scroll bar in order for the result to be displayed.

It is important to note that when a CHANGING event is triggered, the Enhanced Scroll Bar still has the focus. The triggereing of the CHANGING event simply allows you to take a 'time out' to update whatever is being scrolled. Having the result of the scroll being dynamically updated on the screen also makes it easier for the user to produce the exact right setting via scroll bars in one focus session.

Copyright (C) InfoSoft, 1991, 1992

5. Keyboard Status Control (KBStat)

äÒÈ&™*(?&“öPBrush
0 0 82 37àBMÒvR%\
ÄÄ€€€€€€€€€€
€ÀÀÀÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿˆˆˆÿÿ
ÿÿ333ÿ
ÿÿÿÿÿwwwÿÿÿðÿÿÿÿîîîÿÿÿðÿÿÿÿÌÌÌÿÿÿð€ÿÿÿÿ333ÿÿÿð

The Keyboard Status Control (KBSTAT) allows you to paste a display board onto your form that will dynamically reflect the status of the key board locks (Caps, Nums, Insert).

The design was based off the WORD for WINDOWS status bar, since it was well done, well positioned and aesthetically appealing. In highly input intensive applications, a keyboard status bar such as this can be of great aid to the user, especially for touch typists who look at the screen far more than the keyboard.

The KBStat control can be of any size and any location on your form. Too small of size however, and the contents of the control (the three indicator boxes) can be hard to read or chopped.

KBStat is very, very Windows aware. Once installed, it periodically polls the WIN API to check on the status of the KB lock states. If changed, the new status will be updated and displayed. Like WINWORD, the indicator boxes are blank if off, or display "Caps", "Num" and "Ins" when each are ON.

KBStat goes to quite some trouble to find a suitable font for usage based on the size of the control as the capabilities of the display device. Later implementations of KBStat will likely allow for FONTNAME properties, but for now, part of the charm of KBStat is is ease of use and universal compatibility.

In the area of FORECOLOR and BACKCOLOR, the default is black on grey. Naturally, these can be changed via the properties box, but some combinations will look odd on KBStat controls. The thing to keep in mind is that the FORECOLOR is that of the text of the indicators, and that is it. BACKCOLOR is the background color of the parent box and indicators themselves, while the shading for 3D effects is always black, grey and white.

Properties List

FORECOLOR	WIDTH	PARENT
BACKCOLOR	HEIGHT	DRAGMODE
LEFT	VISIBLE	DRAGICON
TOP	ENABLED	TAG

(Custom Properties)
STYLE

Copyright (C) InfoSoft, 1991, 1992

STYLE (Custom Property) (ENUMERATED)

KBStat supports 3 style modes: Convex, Concave and Flat.

In the Convex style, the control appears to be raised above the rest of the form with the indicators sunken into that control or parent box. By far this is the most appealing.

Concave has the control or parent box appearing to be sunken into the form, with the indicators raised above the parent box.

Flat style is flat. Very flat. Neither the form nor the status indicators are accented.

The terms concave and canvex refer to the appearance of the larger parent box, with the smaller indicator boxes or frames being the inverse.

The Style property is read only at run time. Precluding the user or your code from changing the style of the KBStat control allows it to draw the special effects only once rather than each time the status is updated thereby creating less overhead.

Copyright (C) InfoSoft, 1991, 1992

Acknowledgements

The following have proved valuable in one way or another either in designing specific routines or as a general, MASM, system or QB reference:

The Microsoft KnowledgeBase

Norton Guides For Assembler

Copyright (C) InfoSoft, 1991, 1992