Windows Api Examples

Petur J. Skulason.
<mailto:pjs@nett.is?subject=Windows Api Examples>
http://www.est.is/~pjs/RealBasic/
http://u225.nett.is/~pjs/

All this is free to use in any application (commercial - shareware - freeware). Also to extract examples and post as examples from other websites. (I do not take any responsibility for how good this is, I dont call for any credit mony or anythink else ...)

These are few examples that I have found on lists, and by experimenting with Windows API calls and browsing MSDN website (http://msdn.microsoft.com/).

Many text's are copyed from MSDN website in this document are from MSDN website.

Examples are tested on Win98 system, HP OmniBook XE2, compiled on PowerBook G3, PPC7300, LC475. Some of examples are working correctly.

Most of examples are working, but some are not working, possible that i'am doing somthing wrong, but they are there.

Comments, bugs, other samples welcome.
Even requests for more samples. I might try to look at it and see if its possible.

PS:
In this document I have put some links to MSDN webpage's. If you are running Internet Explorer then you might need to 'disconnect java script' couse then ie might show only part of page.
Either run ie with *.js 'OFF' or use an other browser, like iCab or Netscape. (As of aug 30, 2000 then pages are displayed correct... problem might be solved...)

When declaring functions/subs then one is calling library, like user32.dll . Can omitt the name extension, use like Lib "user32" instead of Lib "user32.dll".

Further info:

Argument Data Types

http://msdn.microsoft.com/library/officedev/odeopg/deovrargumentdatatypes.htm

The data types used in C/C++, and the notation used to describe them, differ from those used in VBA. The following table describes some of the common data types used in DLL functions and their VBA equivalents. This list is not all-inclusive, so if you encounter a data type not described here, check one of the reference sources listed in "http://msdn.microsoft.com/library/officedev/odeopg/deovrwheretogofromherech10.htm".

Although you should be familiar with these data types and their prefixes, the Win32API.txt file mentioned earlier contains **Declare** statements ready for use in VBA. If you use these **Declare** statements in your code, the function arguments are already defined with the correct VBA data types.

For the most part, as long as you've defined and passed the correct data types, calling DLL functions works the same way as calling VBA functions. The exceptions are discussed in the following sections.

C/C++ data	**type**	**Hungarian prefix**	**Description**	**VBA equivalent**
BOOL	b	8-bit	Boolean value. Zero indicates False; nonzero indicates True.	Boolean or Long
BYTE	ch	8-bit	unsigned integer	Byte
HANDLE	h	32-bit	unsigned integer that represents a handle to a Windows object	Long
int	n	16-bit	signed integer	Integer
long	l	32-bit	signed integer	Long
LP	lp	32-bit	long pointer to a C/C++ structure, string, function, or other data in memory	Long
LPZSTR	lpsz	32-bit	long pointer to a C-type null-terminated string	Long

Declaring a DLL Procedure

http://msdn.microsoft.com/library/devprods/vs6/vbasic/vbcon98/vbcondeclaringdllprocedure.htm

<quote> (from MSDN website, see link abowe).

> Even though Visual Basic provides a broad set of predefined declares in the Win32api.txt file, sooner or later you'll want to know how to write them yourself. You might want to access procedures from DLLs written in other languages, for example, or rewrite Visual Basic's predefined declares to fit your own requirements.
>
> To declare a DLL procedure, you add a Declare statement to the Declarations section of the code window. If the procedure returns a value, write the declare as a Function:
>
> **Declare Function** *publicname* **Lib** "*libname*" **[Alias** "*alias*"**]** **[([[ByVal]** *variable* **[As** *type***]** **[,[ByVal]** *variable* **[As** *type***]]...])]** **As Type**
>
> If a procedure does not return a value, write the declare as a Sub:
>
> **Declare Sub** *publicname* **Lib** "*libname*" **[Alias** "*alias*"**]** **[([[ByVal]** *variable* **[As** *type***]** **[,[ByVal]** *variable* **[As** *type***]]...])]**
>
> DLL procedures declared in standard modules are public by default and can be called from anywhere in your application. DLL procedures declared in any other type of module are private to that module, and you must identify them as such by preceding the declaration with the Private keyword.
>
> Procedure names are case-sensitive in 32-bit versions of Visual Basic. In previous, 16-bit versions, procedure names were not case-sensitive.
>
> **For More Information** See "Declare Statement" in the *Language Reference*.

Specifying the Library

The Lib clause in the Declare statement tells Visual Basic where to find the .dll file that contains the procedure. When you're referencing one of the core Windows libraries (User32, Kernel32, or GDI32), you don't need to include the file name extension:

```
Declare Function GetTickCount Lib "kernel32" Alias _
"GetTickCount" () As Long
```

For other DLLs, the Lib clause is a file specification that can include a path:

```
Declare Function lzCopy Lib "c:\windows\lzexpand.dll" _
(ByVal S As Integer,    ByVal D As Integer) As Long
```

If you do not specify a path for *libname*, Visual Basic will search for the file in the following order:

5. Directory containing the .exe file
6. Current directory
7. Windows system directory (often but not necessarily \Windows\System)
8. Windows directory (not necessarily \Windows)
9. Path environment variable

The following table lists the common operating environment library files.

Dynamic Link Library	Description
Advapi32.dll	Advanced API services library supporting numerous APIs
Comdlg32.dll	including many security and Registry calls Common dialog API library
Gdi32.dll	Graphics Device Interface API library
Kernel32.dll	Core Windows 32-bit base API support
Lz32.dll	32-bit compression routines
Mpr.dll	Multiple Provider Router library
Netapi32.dll	32-bit Network API library
Shell32.dll	32-bit Shell API library
User32.dll	Library for user interface routines
Version.dll	Version library
Winmm.dll	Windows multimedia library
Winspool.drv	Print spooler interface that contains the print spooler API calls

Working with Windows API Procedures that Use Strings

When working with Windows API procedures that use strings, you'll need to add an Alias clause to your declare statements to specify the correct character set. Windows API functions that contain strings actually exist in two formats: ANSI and Unicode. In the Windows header files, therefore, you'll get both ANSI and Unicode versions of each function that contains a string.

For example, following are the two C-language descriptions for the SetWindowText function. You'll note that the first description defines the function as SetWindowTextA, where the trailing "A" identifies it as an ANSI function:

```
WINUSERAPI
BOOL
WINAPI
SetWindowTextA(
   HWND hWnd,
   LPCSTR lpString);
```

The second description defines it as SetWindowTextW, where the trailing "W" identifies it as a wide, or Unicode function:

```
WINUSERAPI
BOOL
WINAPI
SetWindowTextW(
   HWND hWnd,
   LPCWSTR lpString);
```

Because neither function is actually named "SetWindowText," you need to add an Alias clause to the declare to point to the function you want to reference:

```
Private Declare Function SetWindowText Lib "user32" _
Alias "SetWindowTextA" (ByVal hwnd As Long, ByVal _
lpString As String) As Long
```

Note that the string that follows the Alias clause must be the true, case-sensitive name of the procedure.

Important For API functions you use in Visual Basic, you should specify the ANSI version of a function, because Unicode versions are only supported by Windows NT — not Windows 95/98. Use the Unicode versions only if you can be certain that your applications will be run only on Windows NT-based systems.

Passing Arguments by Value or by Reference

By default, Visual Basic passes all arguments *by reference*. This means that instead of passing the actual value of the argument, Visual Basic passes a 32-bit address where the value is stored. Although you do not need to include the ByRef keyword in your Declare statements, you may want to do so to document how the data is passed.

Many DLL procedures expect an argument to be passed *by value*. This means they expect the actual value, instead of its memory location. If you pass an argument by reference to a procedure that expects an argument passed by value, the procedure receives incorrect data and fails to work properly.

To pass an argument by value, place the ByVal keyword in front of the argument declaration in the Declare statement. For example, the InvertRect procedure accepts its first argument by value and its second by reference:

```
Declare Function InvertRect Lib "user32" Alias _
"InvertRectA" (ByVal hdc As Long, _
lpRect As RECT) As Long
```

You can also use the ByVal keyword when you call the procedure.

Note When you're looking at DLL procedure documentation that uses C language syntax, remember that C passes all arguments except arrays by value.

String arguments are a special case. Passing a string by value means you are passing the address of the first data byte in the string; passing a string by reference means you are passing the memory address where another address is stored; the second address actually refers to the first data byte of the string. How you determine which approach to use is explained in the topic "Passing Strings to a DLL Procedure" later in this chapter.

Nonstandard Names

Occasionally, a DLL procedure has a name that is not a legal identifier. It might have an invalid character (such as a hyphen), or the name might be the same as a Visual Basic keyword (such as GetObject). When this is the case, use the Alias keyword to specify the illegal procedure name.

For example, some procedures in the operating environment DLLs begin with an underscore character. While you can use an underscore in a Visual Basic identifier, you cannot begin an identifier with an underscore. To use one of these procedures, you first declare the function with a legal name, then use the Alias clause to reference the procedure's real name:

```
Declare Function lopen Lib "kernel32" Alias "_lopen" _
(ByVal lpPathName As String, ByVal iReadWrite _
As Long) As Long
```

In this example, `lopen` becomes the name of the procedure referred to in your Visual Basic procedures. The name `_lopen` is the name recognized in the DLL.

You can also use the Alias clause to change a procedure name whenever it's convenient. If you do substitute your own names for procedures (such as using `WinDir` for `GetWindowsDirectoryA`), make sure that you thoroughly document the changes so that your code can be maintained at a later date.

Using Ordinal Numbers to Identify DLL Procedures

In addition to a name, all DLL procedures can be identified by an *ordinal number* that specifies the procedure in the DLL. Some DLLs do not include the names of their procedures and require you to use ordinal numbers when declaring the procedures they contain. Using an ordinal number consumes less memory in your finished application and is slightly faster than identifying a procedure in a DLL by name.

Important The ordinal number for a specific API will be different with different operating systems. For example, the ordinal value for GetWindowsDirectory is 432 under Windows 95 (or later), but changes to 338 under Window NT 4.0. In sum, if you expect your applications to be run under different operating systems, don't use ordinal numbers to identify API procedures. This approach can still be useful when used with procedures that are not APIs, or when used in applications that have a very controlled distribution.

To declare a DLL procedure by ordinal number, use the Alias clause with a string containing the number sign character (#) and the ordinal number of the procedure. For example, the ordinal number of the GetWindowsDirectory function has the value 432 in the Windows kernel; you can declare the DLL procedure as follows:

```
Declare Function GetWindowsDirectory Lib "kernel32" _
Alias "#432" (ByVal lpBuffer As String, _
ByVal nSize As Long) As Long
```

Notice that you could specify any valid name for the procedure in this case, because Visual Basic is using the ordinal number to find the procedure in the DLL.

To obtain the ordinal number of a procedure you want to declare, you can use a utility application, such as Dumpbin.exe, to examine the .dll file. (Dumpbin.exe is a utility included with Microsoft Visual C++.) By running Dumpbin on a .dll file, you can extract information such as a list of functions contained within the DLL, their ordinal numbers, and other information about the code.

For More Information For more information on running the Dumpbin utility, refer to the Microsoft Visual C++ documentation.

Flexible Argument Types

Some DLL procedures can accept more than one type of data for the same argument. If you need to pass more than one type of data, declare the argument with As Any to remove type restrictions.

For example, the third argument in the following declare (`lppt As Any`) could be passed as an array of POINT structures, or as a RECT structure, depending upon your needs:

```
Declare Function MapWindowPoints Lib "user32" Alias _
"MapWindowPoints" (ByVal hwndFrom As Long, _
ByVal hwndTo As Long, lppt As Any, _
ByVal cPoints As Long) As Long
```

While the As Any clause offers you flexibility, it also adds risk in that it turns off all type checking. Without type checking, you stand a greater chance of calling the procedure with the wrong type, which can result in a variety of problems, including application failure. Be sure to carefully check the types of all arguments when using As Any.

When you remove type restrictions, Visual Basic assumes the argument is passed by reference. Include ByVal in the actual call to the procedure to pass arguments by value. Strings are passed by value so that a pointer to the string is passed, rather than a pointer to a pointer. This is further discussed in the section "Passing Strings to a DLL Procedure."
</quote>

Somthing from me

How to do:

Well, how do we do all this?
Ok, lets look at few samples. In sample one I'am going to get windows directory.
By searching MSDN online we can find function that is called: GetWindowsDirectory and it looks like this:

```
Declare Function GetWindowsDirectory Lib "kernel32" Alias "GetWindowsDirectoryA"
(ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Well, how do we analyse this one.
lpBuffer: First two characters starts with lp, then it tells us that it meens 'long pointer' wich then tells us that in REALBasic it is declared as MemoryBlock.
nSize: As long tells us that is Integer value as RB Integer is.
Return Long: Same as nSize, RB Integer.

How it is declared in REALBasic:

```
Declare Function GetWindowsDirectory Lib "kernel32" Alias "GetWindowsDirectoryA"
(ByVal lpBuffer As Ptr, ByVal nSize As Integer) As Integer
```

Memory block is is passed by Reference to function and shall resive path to Windows Directory as 0 terminated string and nSize tells maximum lengt of that buffer that I provide to system to fill. This function returns Integer that tells how long returned string is.
RealBasic function then looks like this:

```
Function fGetWindowsDirectory() As String
   dim m as memoryBlock
   dim r,l as integer
   #IF TargetWin32 then  // *1
   Declare Function GetWindowsDirectory Lib "kernel32" Alias "GetWindowsDirectoryA"
(ByVal lpBuffer As Ptr, nSize As Integer) As Integer
   m = newMemoryBlock(255)
   l = 254
   r = GetWindowsDirectory(m, l)
   return =  m.CString(0)
   #ENDIF then  // *1
End Function
```

**1) Note that '#IF' and '#ENDIF'.*

Then in your app you can call this functon as follows:

```
dim t as string
dim f as folderitem
t = fGetWindowsDirectory()
if right(t,1)<>"\" then
t = t + "\"
end if

f = GetFolderItem(t)

// f now points to for example 'C:\WINDOWS\'
```

Similar are functions that return System directory and Temporay directory:

```
Declare Function GetSystemDirectory Lib "kernel32" Alias "GetSystemDirectoryA"
(ByVal lpBuffer As String, ByVal nSize As Long) As Long

Declare Function GetTempPath Lib "kernel32" Alias "GetTempPathA" (ByVal
nBufferLength As Long, ByVal lpBuffer As String) As Long
```

More Complex call

Lets now look at more complex Api call. Lets Select Color with standard Windows ChooseColor Dialog.

Function call is as follows:

```
Declare Function ChooseColor Lib "comdlg32.dll" Alias "ChooseColorA" (pChoosecolor
As CHOOSECOLOR) As Long
```

That 'p' in start of 'pChoosecolor' tells me that this is pointer to structure in memory that I have to fill with data. Then I can call declare this function as follows:

```
Declare Function ChooseColor Lib "comdlg32" Alias "ChooseColorA" (pChooseColor As
Ptr) As Integer
```

But what is diz 'CHOOSECOLOR'?
Ok, CHOOSECOLOR structure tells me about that:

```
Type CHOOSECOLOR
      lStructSize As Long
      hwndOwner As Long
      hInstance As Long
      rgbResult As Long
      lpCustColors As Long
      flags As Long
      lCustData As Long
      lpfnHook As Long
      lpTemplateName As String
End Type
```

Ok, now analyze this sucker:

lStructSize	long value that is has same value as how many bytes my memoryblock is in size.
hwndOwner	Owner or calling window, 'self.winHWND' property. Tells Windows to what window or application ChooseColor dialog should be put on screen and 'Modality' of dialog. If set to hwnd of MDI form then Dialog will appear staggered to MDI form. If set to hwnd of calling window then dialog will appear staggered to calling window. (Best is to use hwnd of calling window.)
hInstance	Is handle to memory block that contains dialog box template. Is ignored unless set by one of members of 'flags'. Is used if one wants to show his own 'nifty' dialog. (In RB ignored.)
rgbResult	If user hits OK button then this member contains that new color. Is also used to 'Init' dialog with color, like if user wants to change color of somthing then old color is selected in dialog. On member of 'flags' has to set to use that method. If that member is not set then 'initially' set color is 'Black'.
lpCustColors	Long Pointer to memory block where custom colors are stored. Ok, this one is long pointer to memoryblock, and rgb colors are stored as long, (4 bytes). Custom colors are 16 in ChooseColor dialog and then we are going to use 4bytes * 16 = 64 as storage for CustomColors. This memory block you allways has to fill, even though you dont allow user to make custom colors, else user will see only random colors in ‘custom color’ table.

Flags Flags are 32 bit long (4 bytes) bit settings of what Common Dialogs library should to with my data. Only first 9 bits used, (ie I count from lowOrder to highOrder.)

Bit pos	Hex Value	Name	What it does	Status in example
1	&h1	CC_RGBINIT	Tells dialog that value in rgbResult as initially selected color.	provided
2	&h2	CC_FULLOPEN	Dialog will display additonal controls so user can create custom colors. If this bit is not set then user must click 'Define Custom Colors' Button.	provided
3	&h4	CC_PREVENTFULLOPEN	Disables the 'Define Custom Colors' button in dialog.	provided
4	&h8	CC_SHOWHELP	Dialog will show help button. (not working)	Ignored
5	&h10	CC_ENABLEHOOK	Hook enabled (Not possible)	Ignored
6	&h20	CC_ENABLETEMPLATE	Enable template member, (not possible)	Ignored
7	&h40	CC_ENABLETEMPLATEHANDLE	Pointer to template (not possible)	Ignored
8	&h80	CC_SOLIDCOLOR	Tells dialog to display only 'solid colors' in set of basic colors.	provided
9	&h100	CC_ANYCOLOR	Tells dialog to display all available colors in set of basic colors.	provided

lpCustData Not used here.... (long pointer to memoryblock)
lpfnHook Not used here.... (long pointer to memoryblock)
lpTemplateName Not used here.... (long pointer to memoryblock)

Ok, now when we have analyzed how the structure is constructed, then we can see that we have to construct 2 memoryblocks, one for that CHOOSECOLOR struct and one for lpCustomColors.
lpCustColors is memoryblock, wich is 64 bytes long, every color takes 4 bytes from memory, so; 4 bytes * 16 colors = 64 bytes
ChooseColor has 9 members, each as long (integer, 4 bytes long each of them). So, ChooseColor struct is 4bytes * 9 = 36 bytes long.

Member name	VB	size	RB Type	Start Pos
lStructSize	long	4	Integer	0
hwndOwner	long	4	Integer	4
hInstance	long	4	Integer	8
rgbResult	long	4	Integer	12
lpCustColors	long	4	Ptr	16
flags	long	4	Integer	20
~~lCustData~~	~~long~~	~~4~~	~~Integer~~	24
~~lpfnHook~~	~~long~~	~~4~~	~~ptr~~	28
~~lpTemplateName~~	~~long~~	~~4~~	~~ptr~~	32
Count of bytes				36

Now, armed with above information:

I'am going to call ChooseColor dialog, have it to select closest mach of that color that I provide, show full open and provide coustom colors.
I provide my init color, and arry of 16 colors + handle to calling window.

```
Function fChooseColor(cIn as Color, hw As Integer, ByRef cCustom() As Color) As
Color
dim colorStruct,lpCustColors as memoryBlock
dim i,x as integer
#if TargetWin32 then
Declare Function ChooseColor Lib "comdlg32" Alias "ChooseColorA" (pChooseColor As
Ptr) As Integer
// declare memory blocks:
lpCustColors = newmemoryBlock(64)
colorStruct= newMemoryBlock(36)
```

Ok, now I'am going to fill that pChooseColor struct with my data.
First of all is the size of struct:

```
colorStruct.long(0)=36  // 36 bytes long memoryblock
colorStruct.long(4) = hw // Owner window (makes dialog modal to app/window)
colorStruct.long(12) = ColorToInt(cIn)// convert init color to integer (see later)
colorStruct.PTR(16) = lpCustColors // pointer to custom color arry
 // ChooseColor dialog flags, see table abowe
i = &h100// CC_ANYCOLOR, use any color
i=i+&h1  // CC_RGBINIT = &H1 select closest match
i=i+&h2  // CC_FULLOPEN = &h2 Show additonal controls so user can make custom colors
i=i+&h80 // CC_SolidColor Display only solid colors in set of basic colors
colorStruct.long(20) = i // dialog flags
// ** fill custom color arry
for x = 0 to 15
lpCustColors.long(x*4) = ColorToInt(cCustom(x))
next
// now, ready to call dialog
i=ChooseColor(colorStruct) // Call dialog
if i <> 0 then // ChooseColor returns '0' if not OK pressed
for x = 0 to 15
cCustom(x) = IntToColor(lpCustColors.long(x*4)) // extract custom colors
next
Return IntToColor(colorStruct.long(12))
end if
#endif
return cIn // if not selected color... (my incoming color)
End Function
```

Windows rgb color is 4 byte integer, so color can be stored in one integer, last 3 bytes of integer.
Color to Integer:

```
i = c.blue * 65536, for blue part
i=i+c.Green*256, for green part of color
i= i+c.red, for red part of color, note that rgb color 'backwards'
```

And Integer to color is:

```
b = i / 65536
i1 = i - (b*65536)
g = i1 / 256
r = i1 - (g*256)
color = rgb(r,g,b)
```

You can look at color as hex, like $00BBGGRR, then abowe might be simpler to understand.

Examples

ChooseColor

```
Show Standard Windows color picker: this example has custom color arry.
Function fChooseColor(cIn as Color, hw As Integer, ByRef cCustom() As Color) As
Color

dim colorStruct,lpCustColors as memoryBlock
dim i,x as integer
dim ct as color
ct = cIn
#if TargetWin32 then
Declare Function ChooseColor Lib "comdlg32" Alias "ChooseColorA" (pChooseColor As
                  Ptr) As Integer
i = &h100// CC_ANYCOLOR = &H100 any color
i=i+&h1  // CC_RGBINIT = &H1 select closest match
i=i+&h2  // CC_FULLOPEN = &h2 Show additonal controls so user can make custom colors
'i=i+&h4 //CC_PreventFullOpen Does not show custom color button
i=i+&h8  //CC_ShowHelp Show the help button (not working)
i=i+&h80 //CC_SolidColor Display only solid colors in set of basic colors
'can not use together FULLOPEN and PreventFullOpen (will show as fullopen)
lpCustColors = newmemoryBlock(64) // prevent memory corruption *** MUST declare
                                  // else you might get memory coruption, that if
                                  // user adds some custom colors.
colorStruct= newMemoryBlock(36)   // CHOOSECOLOR STRUKT
colorStruct.long(4) = hw // Owner window (makes dialog modal to app/window)
colorStruct.long(12) = ColorToInt(cIn)// convert init color to integer
colorStruct.PTR(16) = lpCustColors // pointer to custom color arry
colorStruct.long(20) = i // dialog flags
colorStruct.long(0)=36 // length of struct // integer pointer to struct
for x = 0 to 15
lpCustColors.long(x*4) = ColorToInt(cCustom(x))
next
i=ChooseColor(colorStruct) // pass that pointer to lib
if i <> 0 then // ChooseColor returns '0' if not OK pressed
for x = 0 to 15
cCustom(x) = IntToColor(lpCustColors.long(x*4))
next
Return IntToColor(colorStruct.long(12))
end if
#endif
return ct // if not selected color... (my incoming color)
End Function
```

Comments about hwnd owner:
if hwnd owner is the MDI window then choose color dialog is ‘staggered’ with MDI window, if hwnd is calling window then dialog is ‘staggered’ with calling window.

```
typedef Struct {                                    size   pos    type
lStructSize      //length in bytes of struct size   4      0      (long)
hwndOwner        //identify owner, can be NULL      4      4      (long)
hInstance        //dialog template                  4      8      (long)
rgbResult        //init color & result color        4      12     (Long)
lpCustColors     //                                 4 (ptr) 16    (long)
Flags            //                                 4       20    (long)
lCustData        //                                 4 (ptr) 24    (long)
lpfnHook         //                                 4 (ptr) 28    (long)
lpTemplateName   //                                 4 (ptr) 32    (long)
}CHOOSECOLOR == 36 bytes
long = 4
```

Further info at:
http://support.microsoft.com/support/kb/articles/Q153/9/29.asp
http://msdn.microsoft.com/library/wcedoc/wcesdkr/ks_ac_23.htm

ChooseColor support functions:

```
Function ColorToInt(c As Color) As Integer
dim i as integer
i = c.blue * 65536
i=i+c.Green*256
i= i+c.red
'msgBox hex(c.red) + ", " + hex(c.green) + ", " + hex(c.blue)
'msgBox right("000000" + hex(i),6)
'strange, but this integer is as b,g,r ,, not as usual r,g,b
return i
End Function

Function IntToColor(i As Integer) As Color
dim c as color
dim r,g,b as integer
dim i1,i2 as integer
b = i / 65536
i1 = i - (b*65536)
g = i1 / 256
r = i1 - (g*256)
'msgBox hex(r) + ", " + hex(g) + ", " + hex(b)
return rgb(r,g,b)

'strange, but this integer is as b,g,r ,, not as usual r,g,b
End Function
```

ExitWindowEx

Shutdown, restart computer or Windows

```
Sub fExitWindowsEx()

dim i1,i2,r as integer#if targetwin32 then
declare function ExitWindowsEx lib "user32.dll" (uFlags As integer, dwReserved As
Integer) As Integer
i1 = 2
i2 = 0
r = ExitWindowsEx(i1,i2)
if r<>0 then
fGetLastError()
end if
#endif
boolean ExitWindowsEx(uFlags (UINT),dwReserved (Ignored))
uFlags:
EWX_LogOff Shut down all processes running in the securiti context of the process
that called the ExitWindowsEx Function. Then logs user off.

EWX_PowerOff Shut down the system and turns off the power. Then logs the user off.
Windows NT must have the SE_SHUTDOWN_NAME priviledge.

EWX_Reboot Shuts down the system and then restarts the system
NT: See abowe

EWX_Shutdown Shuts down the system to taht point at which it is safe to turn off
power. All file buffers have been flushed to disk, and all running processes have
stopped.

EWX_Force Forces processes to terminate. When this flag is set the system does not
send the WM_QUERYENDSESSION AND WM_ENDSESSION messages. This can cause application
to lose data. (do not use)
End Sub
4 = EWX_Force
0 = EWX_Logoff
2 = EWX_Reboot
1 = EWX_shutdown
```

http://msdn.microsoft.com/library/psdk/sysmgmt/shutdown_3ago.htm

GetLastError

Should call LastError immeditly after error, else error might get lost.

```
Sub fGetLastError()
dim r as integer
#if targetwin32 then
Declare function GetLastError Lib "kernel32.dll" Alias "GetLastError" () As Integer
r = GetLastError()
msgBox str(r)
#endif
End Sub
```

http://msdn.microsoft.com/library/wcedoc/wcesdkr/kf_g_19.htm

http://msdn.microsoft.com/library/officedev/odeopg/deovrretrievingerrorinformationfollowingdllfunctioncalls.htm

GetWindowRect

For example use to get size of MDI form. (After call to GetForegroundWindow at application startup)

```
CGetWindowRect (Class)
Properties:
iBottom As Integer
iLeft As Integer
iRight As Integer
iTop As Integer

Sub CGetWindowRect(hw As Integer)
dim i as integer
dim r as memoryBlock
#if targetwin32 then
Declare Function GetWindowRect Lib "user32.dll" (hwnd As Integer, ipRect As Ptr) As
Integer
r = newmemoryBlock(16)
i=GetWindowRect(hw,r)
iLeft = r.long(0)
iTop = r.Long(4)
iRight = r.Long(8)
iBottom = r.Long(12)
#endif
End Sub
```

http://msdn.microsoft.com/library/psdk/winui/windows_471w.htm

GetForegroundWindow

Returns winHWND property of front most window, the window that user is currently working in.(not nessery same as calling application.) Useful to get handle of MDI form at startup.

```
Function fGetForegroundWindow() As Integer
#if targetwin32 then
declare function GetForegroundWindow Lib "user32.dll" Alias "GetForegroundWindow" As Integer
Return GetForegroundWindow()
#endif
End Function

cApp.Open:
Sub Open()
dim ih,r as integer
dim sw,sh as integer
dim b as boolean
dim f as folderItem
dim st as string
dim ss as textoutputStream
// Start with getting handle to mdi
// must be one of first calls before doing anything else
// else you might get handle to other windows
ih = fGetForegroundWindow()
iMDIHandle = ih
sw = screen(0).width
sh = screen(0).height
if ih <>0 then
fMoveWindow(ih,10,10,sw-20,sh-20) //position MDI form as I want it  :)
end if

End Sub
```

http://msdn.microsoft.com/library/psdk/winui/windows_4f5j.htm

Ok, what do we have here..
We are looking for 'handle' to MDI form. There is no need to do that with other windows in rb made apps, couse that handle is alredy there.... as WinHWND, property of window class.
If you want handle to MDI window then call this function at application startup, before you show other windows, like '*default window*' or '*StartupWindow*'

sample:
App.Open event:

```
sub Open
dim... etccc...
iMDIHandle =fGetForegroundWindow()     // iMDIHandle is app.property
WindowStatupShow()
ReadPrefs()
oherStuff()
etc...
end sub

or:
sub Open
dim etcc..
iMDIHandle =fGetForegroundWindow()     // iMDIHandle is app.property
wStartup.Show  // Now is ok to show Startup Window.
ReadPrefs()
PositionMDIForm()
ShowDefaultWindow()
end Sub
```

GetScreenSize

Get the active area of screen, the area abowe / under taskbar, that if taskbar has ‘Auto Hide’ set to false. (The active screen area.)

```
CGetScreenSize (Class)
Properties:
iBottom As Integer
iLeft As Integer
iRight As Integer
iTop As Integer

Sub CGetScreenSize()
dim r as MemoryBlockdim i as integer
#if TargetWin32 then
Declare Function SystemParametersInfo Lib "user32" Alias "SystemParametersInfoA"
(ByVal uAction As Integer, ByVal
r = NewMemoryBlock(16)
i = SystemParametersInfo(48,0,r,0)
iLeft = r.long(0)
iTop = r.long(4)
iRight = r.long(8)
iBottom = r.long(12)

#endif
End Sub
```

http://msdn.microsoft.com/library/periodic/period99/ivb0799.htm

SetWindowText

Set caption of window. Use on MDI caption. No need to to for other windows than MDI form.

```
Function fSetWindowText(hw As Integer, sCaption As String) As Boolean
dim m1 As memoryBlock
dim r as integer
#IF TargetWin32 then
Declare Function SetWindowText Lib "user32" Alias "SetWindowTextA" (ByVal hw As
Integer, ByVal lpString As Ptr) As Integer
m1 = newMemoryBlock(len(sCaption)+1)
m1.cString(0) = sCaption
r = SetWindowText(hw,m1)
if r=0 then
//fgetlasterror()
end if
// return r<>0  // none zero if success.
#endif
End Function
```

http://msdn.microsoft.com/library/wcedoc/wcesdkr/uif_s_43.htm

GetWindowText

Get caption of window. No need on RB windows other than MDI caption, or windows from other applications.

```
Function fGetWindowText(hw As Integer) As String
dim m1 As memoryBlock
dim r as integer
dim t,s as string
#IF TargetWin32 then
Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" (hwnd As Integer,
lpString As Ptr, cch As Integer)
'lpString buffer to put string into...
'cch Integer, allowed length of string, make it one char
' shorter that block, couse this is 'CString'

m1 = newMemoryBlock(40)
r = GetWindowText(hw,m1,39)
t = m1.cString(0)

#endIF
if t<>"" then
return t
end
'unfortunitly rb made app can not return this string
'Crashed with error 5. (access violation)
End Function
```

http://msdn.microsoft.com/library/wcedoc/wcesdkr/uif_fg_78.htm
http://support.microsoft.com/support/kb/articles/Q168/7/51.ASP
http://support.microsoft.com/support/kb/articles/Q129/8/52.asp
http://support.microsoft.com/support/kb/articles/Q183/0/09.ASP

MessageBox

```
mMessageBox.fMessageBox:
Function fMessageBox(hw As Integer, sMsg As String, sCaption As String, iStyle As
Integer) As Integer
dim r as integer
dim lpText, lpCTStr As MemoryBlock
dim uTypex As Integer
#IF TargetWin32 then
Declare Function MessageBox Lib "user32.dll" Alias "MessageBoxA" (hwnd As Integer,
lpText As Ptr, lpCaption As Ptr,wType As Integer) As Integer

lpText = newMemoryBlock(len(sMsg)+1) // cString block 1 byte longer than msg
lpCTStr = NewMemoryBlock(len(sCaption)+1)
lpText.CString(0) = smsg // set message into memory block
lpCTStr.CString(0) = sCaption // set message caption
'utypex = &h3 // Buttons; Yes, No, Cancel
'uTypex=uTypex+&h10 //Icon Stop'uTypex=uTypex+&h200 //Default button (3)
'uTypex=uTypex+&h10000 //Set foreground...
Return MessageBox(hw,lpText,lpCTStr,iStyle)
#ENDIF
End Function
//// set iStyle to some of following constants....
'uFlags; 'mb_' (Button Captions)
'&h2 abortretryignore
'&h0 ok
'&h1 okCancel
'&h4 YesNo
'&h3 YesNoCancel
'&h5 retryCancel
'ICONS;
'&h30    iconExclamation
'        iconWarning
'        iconInformation
'&h40    iconAsterisk
'&h20    iconQuestion
'&h10    iconStop
'        iconError
'        iconHand
'Default buttons;
'&h0     defbutton1
'&h100   defbutton2
'&h200   defbutton3
'&H???   defbutton4
'modality;
'&h0             applmodal
'&h10000         systemmodal
'&h2000          taskmodal
'
'Additonal flags;
'&h20000         default_desktop_only
'help
'right   (direction of text etcc....)
'trlreading      (direction of text etcc....)
'&h10000         setforeground
'topmost
'service_notification
```

http://msdn.microsoft.com/library/wcedoc/wcesdkr/uif_mo_5.htm
http://msdn.microsoft.com/library/devprods/vs6/visualc/vcmfc/_mfc_message.2d.box_styles.htm

ShowWindow

To show windows normal, maximized or iconized.
Useful at application start, or when opening windows to set previus state of window.

```
Function fShowWindow(hw as integer, inState As Integer) As Integer
dim h,i as integer
'sw_showmaximized = 3
'sw_showminimized = 2
'sw_shownormal = 1 //restore
#if targetwin32 then
declare function ShowWindow Lib "user32.dll" Alias "ShowWindow"(ByVal hwnd as
Integer, ByVal nCmdShow as Integer) as Integer
h=hw
i=inState
Return ShowWindow(h,i)
#endif
End Function
```

http://support.microsoft.com/support/kb/articles/Q210/0/90.ASP
http://support.microsoft.com/support/kb/articles/Q89/5/97.ASP

DiskFreeSpace

Use to check for free space on disk. Can be modifyed to show disk total size.

```
Function fDiskFreeSpace(sVol As String) As Integer
dim r,x as integer
dim m,mptr,mx(3) as memoryBlock
dim v As string
'CString lpRootPathName Root directory of disk to check
'long lpSectorsPerCluster Number of sectors per cluster
'long lpBytesPerSector Bytes per sector
'long lpNumbersOffFreeClusters Number of free clusters
'long lpTootalNumberOfClusters Total number of clusters
v = svol + str(r)
#if TargetWin32 then
Declare function GetDiskFreeSpace lib "kernel32.dll" alias "GetDiskFreeSpaceA"
(lpRootPathName As Ptr, lpSectorsPerCluster As Ptr, lpNumbersOffFreeClusters As Ptr,
lpTootalNumberOfClusters As Ptr) As Integer
v = Left(sVol,1) + ":\" // for ex: 'C:\'
m = newMemoryBlock(len(v)+1)
m.cString(0) = v
for r = 0 to 3
mx(r) = newMemoryBlock(4)
next
r = GetDiskFreeSpace(m,mx(0),mx(1),mx(2),mx(3))
// r <> 0 if success
'iSectorsPerCluster = mx(0).Long(0)
'iBytesPerSector = mx(1).Long(0)
'iNumbersOfFreeClusters = mx(2).long(0)
'iTootalNumberOfClusters = mx(3).long(0)
Return mx(0).Long(0)*mx(1).Long(0)*mx(2).long(0) // disk free space
'Return mx(0).Long(0)*mx(1).Long(0)*mx(3).long(0) // disk total size...
#endif
End Function
```

MoveWindow

Use to move windows around. No use on other windows than MDI form in RB.

```
Sub fMoveWindow(hw As Integer, iLeft As Integer, iTop As Integer, iWidth As Integer,
iHeight As Integer)
dim r as integer
#if TargetWin32 then
Declare Function MoveWindow Lib "user32" Alias "MoveWindow" (ByVal hwnd As Integer,
x As Integer, ByVal y As Integer, ByVal bRepaint As Integer) As Integer

r = MoveWindow(hw,iLeft, iTop, iWidth, iHeight, &hF)
if r<> 0 then
end
'return r<>0
#endif
End Sub
```

http://msdn.microsoft.com/library/wcedoc/wcesdkr/uif_mo_5.htm

GetSystemInfo

CGetSystemInfo (Class)

Properties

idwOemID As Integer
idwPageSize As Integer
ilpMiniumApplicationAddress As Integer
lpMaxiumApplicationAddress As Integer
idwActiveProcesorMask As Integer
idwNumberOfProcessors As Integer
idwProcessorType As Integer
idwAllocationGranularity As Integer
iwProcessorLevel As Integer

```
CGetSystemInfo.CGetSystemInfo:
Sub CGetSystemInfo()
dim m as memoryBlock
dim r as integer
#IF TargetWin32 then
Declare Sub GetSystemInfo Lib "kernel32" Alias "GetSystemInfo" (lpSystemInfo As Ptr)
//
m = newMemoryBlock(36)
GetSystemInfo m
idwOemID = m.long(0)
idwPageSize = m.long(4)
ilpMiniumApplicationAddress = m.long(8)
lpMaxiumApplicationAddress = m.long(12)
idwActiveProcesorMask = m.long(16)
idwNumberOfProcessors = m.long(20)
idwProcessorType = m.long(24)
idwAllocationGranularity = m.long(28)
iwProcessorLevel = m.long(32)
#endif
'SYSTEM_INFO STRUCT:
'dwOemID         As Long (4 bytes) //absoulte, used on NT sytem prior to 3.51
'dwPageSize      As Long (4 bytes)
'lpMiniumApplicationAddress      As Long (4 bytes)
'lpMaxiumApplicationAddress      As Long (4 bytes)
'dwActiveProcesorMask            As Long (4 bytes)
'dwNumberOfProcessors            As Long (4 bytes)
'dwProcessorType                 As Long (4 bytes)
'dwAllocationGranularity         As Long (4 bytes)
'wProcessorLevel { struct (see links below)
'
' Total: 36 bytes
End Sub
```

http://msdn.microsoft.com/library/wcedoc/wcesdkr/ks_qs_16.htm
http://msdn.microsoft.com/library/wcedoc/wcesdkr/kf_g_35.htm
http://msdn.microsoft.com/library/psdk/sysmgmt/sysinfo_5r76.htm
http://msdn.microsoft.com/library/psdk/sysmgmt/sysinfo_8stv.htm

From MSDN website:

wProcessorArchitecture

Specifies the system's processor architecture. It is no longer relevant. Use the *wProcessorArchitecture*, *wProcessorLevel*, and *wProcessorRevision* members to determine the type of processor.

* PROCESSOR_ARCHITECTURE_INTEL
* PROCESSOR_ARCHITECTURE_MIPS
* PROCESSOR_ARCHITECTURE_ALPHA
* PROCESSOR_ARCHITECTURE_PPC
* PROCESSOR_ARCHITECTURE_UNKNOWN

dwActiveProcessorMask

Specifies a mask representing the set of processors configured into the system. Bit 0 is processor 0; bit 31 is processor 31.

dwNumberOfProcessors

Specifies the number of processors in the system.

dwProcessorType

Specifies the type of processor in the system. This member is no longer relevant. Use the *wProcessorArchitecture*, *wProcessorLevel*, and *wProcessorRevision* members to determine the type of processor.

This member is one of the following values:

* PROCESSOR_INTEL_386
* PROCESSOR_INTEL_486
* PROCESSOR_INTEL_PENTIUM
* PROCESSOR_INTEL_486
* PROCESSOR_MIPS_R3000
* PROCESSOR_MIPS_R4000
* PROCESSOR_HITACHI_SH3
* PROCESSOR_HITACHI_SH4
* PROCESSOR_PPC_403
* PROCESSOR_PPC_821
* PROCESSOR_STRONGARM
* PROCESSOR_ARM720

wProcessorLevel

Windows NT/2000: Specifies the system's architecture-dependent processor level. If **wProcessorArchitecture** is PROCESSOR_ARCHITECTURE_INTEL, **wProcessorLevel** can be one of the following values.

Value	Meaning
3	Intel 80386
4	Intel 80486
5	Intel Pentium
6	Intel Pentium Pro or Pentium II

If **wProcessorArchitecture** is PROCESSOR_ARCHITECTURE_MIPS, **wProcessorLevel** is of the form 00*xx*, where *xx* is an 8-bit implementation number (bits 8-15 of the PRId register). The member can be the following value.

Value	Meaning
0004	MIPS R4000

If **wProcessorArchitecture** is PROCESSOR_ARCHITECTURE_ALPHA, **wProcessorLevel** is of the form *xxxx*, where *xxxx* is a 16-bit processor version number (the low-order 16 bits of a version number from the firmware). The member can be one of the following values.

Value	Meaning
21064	Alpha 21064
21066	Alpha 21066
21164	Alpha 21164

If **wProcessorArchitecture** is PROCESSOR_ARCHITECTURE_PPC, **wProcessorLevel** is of the form *xxxx*, where *xxxx* is a 16-bit processor version number (the high-order 16 bits of the Processor Version Register). The member can be one of the following values.

Value	Meaning
1	PPC 601
3	PPC 603
4	PPC 604
6	PPC 603+
9	PPC 604+
20	PPC 620

wProcessorRevision
Windows NT/2000: Specifies an architecture-dependent processor revision. The following table shows how the revision value is assembled for each type of processor architecture.

Processor	Value
Intel 80386 or 80486	A value of the form *xxyz*.

If *xx* is equal to 0xFF, *y* - 0xA is the model number, and *z* is the stepping identifier. For example, an Intel 80486-D0 system returns 0xFFD0.

If *xx* is not equal to 0xFF, *xx* + 'A' is the stepping letter and *yz* is the minor stepping.

Intel Pentium, Cyrix, or NextGen 586	A value of the form *xxyy*, where *xx* is the model number and *yy* is the stepping. Display this value of 0x0201 as follows:

Model *xx*, Stepping *yy*

MIPS	A value of the form 00*xx*, where *xx* is the 8-bit revision number of the processor (the low-order 8 bits of the PRId register).
ALPHA	A value of the form *xxyy*, where *xxyy* is the low-order 16 bits of the processor revision number from the firmware. Display this value as follows:

Model A+*xx*, Pass *yy*

PPC	A value of the form *xxyy*, where *xxyy* is the low-order 16 bits of the processor version register.

GetVersionEx

CGetVersionEx (Class)
Properties:

```
idwMajorVersion As Integer
idwMinorVersion As Integer
idwBuildNumber As Integer
sszCSDVersion As String
i9xMinor As Integer
i9xMajor As Integer
idwPlatformId As Integer

CGetVersionEx.CGetVersionEx:
Sub CGetVersionEx()
dim m as memoryBlock
dim r as integer
dim p As Integer
dim t As String
#if targetwin32 then
Declare Function GetVersionEx Lib "kernel32" Alias "GetVersionExA"
(lpVersionInformation As Ptr) As Integer
m = NewMemoryBlock(148)
m.Long(0) = 148
r = GetVersionEx(m)
idwMajorVersion = m.long(4)
idwMinorVersion = m.long(8)
idwBuildNumber = m.long(12)
idwPlatformId = m.long(16)
sszCSDVersion = m.CString(20)
if idwPlatformId=1 then
'Windows 95/98:
'Identifies the build number of the operating system in the low-order word.
'The high-order word contains the major and minor version numbers.
t = right("00000000" + hex(idwBuildNumber),8) //might be simpler way to do this,,
idwMajorVersion = val("&h" + left(t,2))  byte 1 of long
idwMinorVersion = val("&h" + mid(t,3,2)) byte 2 of long
idwBuildNumber = val("&H" + right(t,4))  byte 3&4 of long
end if
#endif
'OSVERSIONINFOEX
'dwOSVersionInfoSize 4 bytes
'dwMajorVersion 4 Bytes //NT
'dwMinorVersion 4 Bytes //NT
'dwBuildNumber 4 Bytes //NT Build NO,
'dwPlatformId 4 Bytes
'szCSDVersion 128 bytes, NULL terminated string (CString)
' Total 148 bytes
'idwPlatformId == 0, Windows 3.1
'idwPlatformId == 1, Windows 95/98
'idwPlatformId == 2, Windows NT/2000
End Sub
```

http://msdn.microsoft.com/library/psdk/sysmgmt/sysinfo_3a0i.htm
http://msdn.microsoft.com/library/psdk/sysmgmt/sysinfo_49iw.htm
http://msdn.microsoft.com/library/psdk/sysmgmt/sysinfo_41bi.htm

GetWindowText

mGetWindowText()

Use to get caption of window, like MDI form, no need to use on other windows where there is alredy access to that information throught ‘windowX.Title’ property.

```
Function fGetWindowText(hw As Integer) As String
   dim m1 As memoryBlock
   dim r as integer
   dim t,s as string

   #IF TargetWin32 then
   Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" (hwnd As
Integer, lpString As Ptr, cch As Integer) As Integer
   'lpString   buffer to put string into...
   'cch        Integer, allowed length of string, make it one char
   '           shorter that block, couse this is 'CString'
   'Return     length of string, no need to memoryblock.CString
   m1 = newMemoryBlock(256)
   r = GetWindowText(hw,m1,255)
   t =  m1.cString(0)
   #endIF
   return t

End Function
```

GetShortPathName

Use to get short name of folderitem.
Like to get Progra~1 name of 'Program Files'.

```
Function fGetShortPathName(sLongPathName As String) As String
    dim lpShort,lpLong As MemoryBlock
    dim iBuff,iLen As Integer
    #IF TargetWin32 then
    //Declare Function GetShortPathName
    //Lib "kernel32" Alias "GetShortPathNameA" (
    //ByVal lpszLongPath As String, --> change to ptr
    //ByVal lpszShortPath As String,--> change to ptr
    //ByVal cchBuffer As Long) --> Integer
    //As Long --> Integer
    // *** look out for line breaks... :
    Declare Function GetShortPathName Lib "kernel32" Alias "GetShortPathNameA"
     (ByVal lpszLongPath As Ptr, ByVal lpszShortPath As Ptr, ByVal cchBuffer As
     Integer) As Integer
    // lpszLongPath == long pointer to block in memory ;
    // lpszShortPath == -- ditto ;
    // cchBuffer == size of resiving block ;
    lpShort = newmemoryBlock(255)
    lpLong = newMemoryBlock(len(sLongPathName)+2)
    iBuff = 254
    if right(sLongPathName,1) <> "\" then
     lpLong.CString(0) = sLongPathName+"\"
    else
     lpLong.CString(0) = sLongPathName
    end if
    iLen = GetShortPathName(lpLong,lpShort,iBuff)
    if iLen <> 0 then
     Return lpShort.CString(0)
    else
     return "### Error Getting GetShortPathName: " + sLongPathName
    end if
    #endif
End Function
```

GetParentWindow

Use to get parent of some window.
Can be used at application startup, use from 'Splash Window'
Note; you have to call two times (se belowe).

```
Function fGetParentWindow(hWindow As Integer) As Integer
#if targetwin32 then
Declare Function GetParent Lib "user32" Alias "GetParent" (ByVal hwin As Integer) As
Integer
dim r,hw As integer
hw = hWindow
r = getparent(hw)
return r
#endif
End Function
```

Well, I don't know what the code looks like, but here's what I would do,
assuming you have an MDI child window (document window).
You can put this in the Open event of the child window if you like:

```
#if TARGETWin32 then
Declare Function GetParent Lib "user32" (hWnd as Integer) as integer
Declare Sub ShowWindow Lib "user32" (hWnd as Integer, cmdShow as Integer)
Dim MDIwnd as Integer

Const SW_HIDE = 0
Const SW_SHOWNORMAL = 1
Const SW_SHOWMINIMIZED = 2
Const SW_SHOWMAXIMIZED = 3

MDIwnd = GetParent(GetParent(self.WinHWND))
ShowWindow MDIwnd, SW_SHOWMAXIMIZED
#endif
```

Note the 2 GetParent calls, the first GetParent call gets the parent of the Child Window (**which IS NOT the MDI application itself, but a hidden MDI client window).** The next GetParent call returns the parent of the MDI client window, which is the MDI application window itself, so now you have the MDI window handle.

Regards,
William Yu
william@realsoftware.com

(Note though; in my tests I dont use that 'double call', becouse that was not working for me –pjs).

GetCommandLine

Use to get commands passed to application when started. Also contains path to its self and dropped folderitems at startup.

Use As:

```
#if targetwin32
Declare Function GetCommandLine Lib "kernel32" Alias "GetCommandLineA" () As Ptr
dim m As memoryBlock
m = GetCommandLine()
Return m.cstring(0)
#endif
```

Samples:

```
Function PathToSelf() As String
   dim s,d2 As string
   dim i1,i2 As integer

   s = fGetCommandLine()
   i1 = instr(s,chr(34))  '   "path to self is inside
                          ' ""
   i2 = instr(i1+1,s,chr(34)) ' find '"' after i
   d2 = mid(s,i1+1,i2-1-i1)

   'msgBox d2
   return d2
End Function
```

```
Sub GetDropedItems(f() As FolderItem)
   dim x,c As integer
   dim s1,s2 As String
   dim i1,i2 As integer
   dim f1 As folderItem

   s1 = fGetCommandLine()

   i1 = instr(s1,chr(34))  '   "
   i2 = instr(i1+1,s1,chr(34)) ' find '"' after i1

   s1 = mid(s1,i2+1) ' rest of string

   if len(s1) > 0 then
      c = countfields(s1," ")
      for x = 1 to c
         s2 = nthField(s1," ",x)
         f1 = getfolderitem(s2)
         if f1.exists then
            f.append f1
         end if
      next
   end if
   'c = ubound(f)
   'for x = 0 to c
   'msgBox f(x).absolutePath
   'next
End Sub
```

Create Shortcuts

Create shortcuts by code in Win32.
To create that shortcut you start by creating shortcut in 'RecentDocs' folder, then you can move/copy to new location from there.
What I have seen, is; does only work for folderitems that system knows of. If file extension is not known then creation of shortcut will fail silently.

Create shortcut in 'Resent docs':

```
Function AddToRecentDocs(dwFlags As Integer, dwData As String) As Integer
   #If targetwin32
   Declare Function api_SHAddToRecentDocs Lib "shell32" Alias "SHAddToRecentDocs"
(ByVal dwFlags As Integer, ByVal dwData as Ptr) As Integer
   dim m As MemoryBlock
   dim r As integer

   m = NewMemoryBlock(len(dwData) + 1)
   m.cstring(0) = dwData
   r = api_SHAddToRecentDocs(dwFlags,m)
   'msgBox str(r)
   if r>0 then // no err
      return r
   else
      FindError()
      return -1
   end if
   #endif
End Function

dwData is absolute path to folderitem to create shortcut for
dwFlags is somthing, use '2' in this case...
```

Nessersary functions:

To get special folders, by system numerics, couse folders can have different names and location pending of system version or language, so better not hardcode path, like for 'Desktop' or other locations:

```
#if targetwin32
  Declare Function api_SHGetSpecialFolderLocation Lib "shell32.dll" Alias
"SHGetSpecialFolderLocation" (ByVal hwndOwner As Integer, ByVal nFolder As Integer,
byVal pidl As Ptr) As Integer
  dim r As integer
  dim iFold As integer
  dim rFolder As integer
  dim rm As memoryBlock

  iFold = ifolder

  rm = newMemoryBlock(5) '4 bytes is enough...

  'msgBox "Calling r = api_SHGetSpecialFolderLocation(hw,iFolder,rFolder)"
  r = api_SHGetSpecialFolderLocation(hw,iFold,rm)
  'msgBox "End Calling"
  if r = 0 then
    rfolder = rm.long(0)
    return rFolder
  else
    msgBox STR(r)
  end if
  #endif
```

And then we need absolute path from this:

```
Function GetPathFromIDList(pidl As Integer) As String
   #if targetWin32
   Declare Function api_SHGetPathFromIDList Lib "shell32" Alias
"SHGetPathFromIDList" (ByVal pidl As Integer, ByValsPath As Ptr) As Integer

   dim m As memoryBlock
   dim r As integer
   dim sPath As string

   m = newMemoryBlock(260) 'or 1024/2048/4096
   r = api_SHGetPathFromIDList(pidl,m)
   if r>0 then// no err
      sPath = m.cstring(0)
      return sPath
   else
      FindError()
      return ""
   end if
   #endif
End Function
```

Ok, And then I can create shortcut and copy it then to location that i want;

```
Function CreateShortcutIn(hw As Integer, fSource As FolderItem, fTargetFolder As
FolderItem, targetName As String) As Integer
   dim SourcePath As String
   dim SourceName As String
   dim Recentpath As String
   dim TargetFolder As String
   dim iRef,iList As Integer
   dim r,i As integer
   dim fCreatedShortcut,f2 As folderItem
   dim t As integer

   if (not fSource.exists) or fSource=nil then
      return -1  // source not valid
   end if
   if (not fTargetFolder.exists) or fTargetFolder=nil then
      return -2  // target not valid
   end if

   SourcePath = fSource.absolutePath
   TargetFolder = fTargetFolder.absolutePath
   SourceName=fSource.name
   'msgBox SourceName
   // Modifycation by Steve Evans
   'i=InStr(fSource.name, ".")
   'if i<>0 then
   'SourceName=left(fSource.name,i-1)
   'else
   'SourceName=fSource.name
   'end if
   //
   // create shortcut for source in 'RecentFolder'
   r = AddToRecentDocs(2,SourcePath)
   if r>-1 then // could create shortcut
      'wait for system to finish task:
      t = ticks
      while ticks-t < 80 // wait for system... 1,3 seconds
         r = r + 1
      wend
```

```
      // Find path to Recentdocs folder
      iRef = GetSpecialFolderLocation(hw,CSIDL_RECENT) 'CSIDL_RECENT == 8
      if iRef>0 then
         Recentpath = GetPathFromIDList(iRef)  // find recentfolder
         if Recentpath<> "" then
            f2 = getFolderItem(Recentpath)
            if f2.exists then
               fCreatedShortcut = f2.child(Sourcename + ".LNK")
               if fCreatedShortcut.exists then
                  fCreatedShortcut.name = targetname + ".LNK"
                  if fTargetFolder.child(fCreatedShortcut.name).exists then
                     // delete old link
                     fTargetFolder.child(fCreatedShortcut.name).delete
                  end if
                  'msgBox fCreatedShortcut.absolutePath
                  'msgBox fTargetFolder.absolutePath
                  fCreatedShortcut.moveFileTo fTargetFolder
               else
                  // Try other mothod:
                  i = len(SourceName)
                  for r = i downTo 1
                     if mid(SourceName,r,1) = "." then
                        SourceName = left(SourceName,i)
                        fCreatedShortcut = f2.child(Sourcename + "LNK")
                        if fCreatedShortcut.exists then
                           fCreatedShortcut.name = targetname + ".LNK"
                           if fTargetFolder.child(fCreatedShortcut.name).exists then
                              // delete old link
                              fTargetFolder.child(fCreatedShortcut.name).delete
                           end if
                           'msgBox fCreatedShortcut.absolutePath
                           'msgBox fTargetFolder.absolutePath
                           fCreatedShortcut.moveFileTo fTargetFolder
                           Return 0 // succes
                        end if
                     end if
                  next
                  return -101 ' created shortcut does not exist or not found..
               end if 'if not fCreatedShortcut.exists then
            else
               return -5 'could not get folderitem of recent docs
            end if 'f2 = getFolderItem(Recentpath)
         else
            Return -4 // could not get GetPathFromIDList
         end if 'if Recentpath<> "" then
      else
         return -3 ' could not find 'RecentDocs' folder
      end if 'iRef>0
   else
      return -11 // ERROR: could not create in recentdocs...
   end if 'r>-1,, r = AddToRecentDocs(2,SourcePath)
End Function
```

Note; that i try move item on two different ways; (if fCreatedShortcut.exists then):
version 1 works of my omnibook, but method 2 did not work, but for someone else then method 1 was not working, but his modifycations where working for him, so i added his modifycations to code, so it might work to.