PRODUCT	Dans Visual Basic Tools
Description	Power Tools for Visual Basic(R) programmers
File Name	VBMAGIC.BAS
Object Type	Visual Basic
Remarks	Windows API wrap-arounds provide enhanced control features
About	Useful tools to enhance product quality and aid fast development

Overview

VBMAGIC.BAS is a collection of useful tools for visual basic programmers. The functions and subroutines provided in this module allow the basic programmer to develop applications with features normally beyond Visual Basic. These routines are packaged as application independent modules and are easily reused in most projects.

This document describes the application programmer interface to the functionality found in VBMAGIC.BAS. Since there is now a commercial product named VB Magic, please do not confuse this file with that product. The two are very much unrelated. VBMAGIC.BAS is periodically updated by the author, and is always available to any interested parties. VBMAGIC.BAS is intended to function with versions of Visual Basic 2.0 and above. Support for version 1 of visual basic is not available any longer.

CONTENTS

Subroutines/function

ADDLISTITEM **add an item to a listbox/combobox with its associated data item**

Syntax:

i% = addlistitem(ctl as control, text$, index%, dataval&)

where:

ctl is a reference to the listbox control being modified
text is the string to be inserted into the control
index is the position (zero based) to perform the insert
dataval is a long integer to be put into the itemdata property

Remarks

This function is a legacy from the VB 1.0 days when a listbox control did not support direct access to its itemdata value array. It is kept in this module so that applications written to VB 1.0 and then converted will continue to function. The functionality provided is now available via VB listbox properties, although more than one line of code is required.

Return Value

This function returns the index value supplied

ADDSLISTITEM **add an item to a SORTED listbox/combobox with its associated data item**

Syntax:

i% = addlistitem(ctl as control, text$, dataval&)

where:

ctl is a reference to the listbox control being modified
text is the string to be inserted into the control
dataval is a long integer to be put into the itemdata property

Remarks

This function is a legacy from the VB 1.0 days when a listbox control did not support direct access to its itemdata value array. It is kept in this module so that applications written to VB 1.0 and then converted will continue to function. The functionality provided is now available via VB listbox properties, although more than one line of code is required.

Return Value

This function returns the zero based index of the position where the new item has been inserted.

BOXTRACK **routine to create a tracked listbox/textbox relationship**

Syntax:

boxtrack textboxref, listboxref

where:

textboxref is a reference (name of) to a textbox control, or any control with a TEXT property.
listboxref is a reference to a listbox (not a combobox or dropdown list)

USE:

To link a list box control to an edit control, add the code below to the KEYPRESS event for the edit box control. This causes each keypress to refresh the list-box.
Example:

```
SUB MYEDITCONTROL_KEYPRESS (keyascii as integer)
    boxtrack myeditcontrol, mylistboxcontrol
END SUB
```

Remarks:

Tracked listboxes are seen in the standard Windows(tm) 3.1 help system under the search feature. The text box above the list box allows a user to type topic names to be located in the list. As the user types each character, the listbox is repositioned to the nearest item. This function allows a programmer to provide tracked list boxes in any application.

CASCADECHILDREN **performs Windows version independent cascade operation on child windows**

Syntax:

cascadechildren parent_hwnd%, style%

where:

Parent_Hwnd is the window handle of the specified "parent window"
Style: Meaningful only in Windows 3.1 and up. Refer to SDK function reference for values.

CENTER **routine to center the specified form on the screen**

Syntax:

center hwnd%

where:

hwnd Window handle of form to be centered
EXAMPLE:

```
center Me       ' centers the current form
center thee     ' centers a form whose name property is set to THEE
```

CRYPT **encrypts and decrypts string data**

Syntax:

s$ = crypt(action$, key$, src$)

where:

s: A string to hold the encrypted/decrypted value
key: String holding action verb (**E**ncrypt or **D**ecrypt): Permissible values: E, D
src: String holding data to be encrypted or decrypted

This function performs key based XOR encryption on string data. The algorithm used is reversible when the same key is used to encrypt and decrypt the string. It is suggested that the key string be fairly random and not match the string being encrypted. Longer keys yield more secure results. The function will cycle through a key whose length is less than the length of the string being acted upon. The result of the operation is returned as the function value.

EM_GETLINETEXT **fundtion that extracts a specific line from an edit box control.**

Syntax:

s$ = em_getlinetext(editcontrol, lineno)

where:

s: A string to hold the return value
editcontrol: the name of an edit control (name property value)
lineno: zero based line position to extract

This useful function extracts a given visual line from a multi-line edit control. This function will work with any control that supports EM_GETLINE messages. The line number is zero based, meaning that the first line in the control work area is referenced with a zero value, the second line as one, etc. Useful for extracting text as the user sees it. This differs from extracting the **text** property value, which ignores line breaks.

EM_GETNUMLINES **function to extract the logical line count from an edit control**

Syntax:

i% = em_getnumlines(editcontrol)

where:

i: An integer to hold the return value
editcontrol: the name of an edit control (name property value)

This function returns the number of logical lines visible in the work area of a multi-line edit control. The number returned will be one for a single line, two for two visible lines, etc.

FINDSTRING **function to position a list-box control to the next line containing a string**

Syntax:

i% = findstring(listctl, start%, srch$)

where:

i: An integer to hold the return value (line number where match occured)
listctl: reference (name property) to a list control
start: logical list line number to start searching at (zero based)
srch: string variable holding the text to be located

This function searches forward in a list box from the starting position specified. Each line is examined much in the same manner as the basic command *INSTR*, in a case-less comparison for a partial string match. The first line found containing the search string (**srch**) will be identified in the return value. The list is not altered in any way during the search. The programmer is responsible for taking the appropriate action. The return value is a zero based list position. Not found is indicated by a return value less than zero.

FINDSTRINGEXACT **function to position a list-box control to the next line matching a string**

Syntax:

i% = findstringexact(listctl, start%, srch$)

where:

i: An integer to hold the return value (line number where match occured)
listctl: reference (name property) to a list control
start: logical list line number to start searching at (zero based)
srch: string variable holding the text to be located (exact match)

This function searches forward in a list box from the starting position specified. Each line is examined for an exact match(whole line), in a case-less comparison. The first line found containing the search string (**srch**) will be identified in the return value. The list is not altered in any way during the search. The programmer is responsible for taking the appropriate action. The return value is a zero based list position. Not found is indicated by a return value less than zero.

GETPROFILEINT **function returns integer value from current application INI file**

Syntax:

i% = getprofileint(sect$, entry$, default%))

where:

i: An integer to hold the return value
sect: section identifier found in INI file . e.g. [SAMPLE]
entry: section entry identifier. e.g. ENTRY= 4
default: value to be returned if sample/entry pair not found

This wrapper function has the SAME NAME as a standard windows API function, but is intended to over-ride the use of the standard call so that the global WIN.INI is left uncluttered. This function works in conjunction with the SETAPPNAME function described under its own heading later on. The SETAPPNAME function allows the programmer to specify the name of an INI file to be used throughout the program until the next call to SETAPPNAME. All calls to the family of INI Set/Get functions found in this function collection require that the set-up function SETAPPNAME be called first.

GETPROFILESTRING **function to extract character values from application INI file**

Syntax:

i% = getprofilestring(sect$, entry$, default$, buffer%, size%)

where:

i: An integer to hold the return value (new length of buffer argument)
sect: section identifier found in INI file . e.g. [SAMPLE]
entry: section entry identifier. e.g. ENTRY= XYZ123
default: text value to be returned if sample/entry pair not found
buffer: name of string variable to be populated with return value
size: maximum number of characters to be returned.

This wrapper function has the SAME NAME as a standard windows API function, but is intended to over-ride the use of the standard call so that the global WIN.INI is left uncluttered. This function works in conjunction with the SETAPPNAME function described under its own heading later on. The SETAPPNAME function allows the programmer to specify the name of an INI file to be used throughout the program until the next call to SETAPPNAME. All calls to the family of INI Set/Get functions found in this function collection require that the set-up function SETAPPNAME be called first.

Unlike the standard Windows API with the same name, no extra care need be taken with the string return value. The wrapper function shields the programmer from the consequences of non-allocated buffer storage. The maximum suggested length for INI entries (Microsoft as source) is 256 characters.

GETSYSTEMDIRECTORY **function returns full path to logical WINDOWS/SYSTEM directory**

Syntax:

s$ = getsystemdirectory()

where:

s: A string to hold the return value

The windows *system* directory is a logical path name where, upon start-up, windows found its main dynamic link library components. On a stand-alone installation of windows, the default system directory path is C:\WINDOWS\SYSTEM. On networks, this varies, and is PATH environment variable dependent. To perform installation type activities, it is useful to have a function that locates the current system directory.

GETVERSION **function returns windows version number as an integer**

Syntax:

i% = getversion()

where:

i: An integer to hold the return value

This function returns the windows version number as: major * 100 + minor. Knowledge of windows minor version numbers help here. Windows version 3.1 will be returned as 310, where version 3.00 will be identified as 300 by this function.

GETWINDOWSDIRECTORY **function returns full path to logical WINDOWS directory**

Syntax:

s$ = getwindowsdirectory()

where:

s: A string to hold the return value

The *windows* directory is a logical path name where, upon start-up, windows found its start-up components (win.com). On a stand-alone installation of windows, the default windows directory path is C:\WINDOWS. On networks, this varies, and is PATH environment variable dependent. To perform installation type activities, it is useful to have a function that locates the current windows directory. This is also the default path where windows looks for INI files.

NOTELAUNCH **routine to run/activate a copy of notepad for edit/view of a specific .TXT**

Syntax:

notelaunch fileroot

where:

fileroot: A string holding the name of a .TXT file.

This subroutine launches or locates and activates a copy of the notepad utility that shipped with Windows, loading a copy of the file specified. After determining the file name, which is forced

by the subroutine to end with .TXT, the windows task list is searched for an existing copy of notepad that is running with the same file loaded. If none is found, a new copy of notepad is started; otherwise the existing copy is brought to the foreground.

Since notepad is a seperate application, there is no further linkage to the program via this routine.

RELATEMETO **routine to link a form to another form as parent/child.**

Syntax:

relatemeto childform, parentform

where:

childform: form name property of form to be child in the new relationship
parentform: form name property of form to be parent in the new relationship

This subroutine allows the programmer to alter the Visual Basic form behavior so that visual basic programs behave more like standard windows programs. Specifically, when the parent-child relationship is established, window related events and commands can be utilized. Some of these are cascading, tiling, minimizing. To see the effect of this function, relate form2 to form1, display both forms, and then minimize form1. The icon for form2 is removed if it was minimized, and the only icon remaining is that of the parent form. Typically this relationship is established in the FORM_LOAD event. A side effect of this function is that child forms need to know the name of the parent form. This conflicts with good object oriented user interface design where reusable application form components should not know the calling form.

SEARCHWINDOW **function to identify a window whose title contains specific text**

Syntax:

i% = searchwindow(stringtofind)

where:

i: An integer to hold the return window handle value
stringtofind: A string holding the text to locate

This function performs a system wide search of all visible windows, selecting the title text from each, and performing a case-insensitive search of the title text for a match on the string to find value. The window handle value (HWND) of the first window whose title contains the search text is returned. If no window is found that matches the search, then zero is returned.

SELECTSTRING **function performs a positioning list box search for an exact match on a search string**

Syntax:

i% = selectstring(listctl, start%, match$)

where:

i: An integer to hold the return value
listctl: a reference to a LISTBOX or COMBOBOX control

start: list control line position to begin searching
match:string variable containing text to locate

Selectstring searches for an exact match in a list control, and if found, repositions the listbox so that the item is visible and selected. Any previous selections are de-selected.

SETAPPNAME **routine to establish application INI working file**

Syntax:

setappname INI_NAME

where:

INI_NAME: A string holding the name of the application INI file

This routine sets the global application INI context for the duration of the application run (or until setappname is run again). Calls to the profile related functions (see getprofilexxx, writeprofilexxx) will reference the INI (initialization constant file) named in the call to this routine.

Using individual INI files for seperate logical applications keeps the global windows initialization file (WIN.INI) free of application specific information. This in turn makes it easy to install copies of applications that may have conflicts in INI section names, as each logical application can see a different view.

As a matter of style, it is considered poor form for an application to require an INI file that it does not create. Suggested practice is for INI values to be built as an application runs. INI entries are very useful for storing last view, options settings, data source information (encrypted is best), and other application specific data. If a suite of applications needs access to common initialization information, they can easily share an INI file via use of SETAPPNAME.

SETEMREADONLY **function to set an edit control to read-only mode**

Syntax:

i% = setemreadonly(ectl, bool%)

where:

i: An integer to hold the return value
ectl: reference (name property value) to an edit control
bool: boolean flag integer value (TRUE/FALSE) determine edit control result status

This function is used to set the read-only status in an edit box. This differs from altering the enabled property in that the control text is left as is, and the control is still able to obtain input focus. While an edit control is in read-only mode, the user may scroll any text, and perform clipboard read operations (cut/copy). The user is unable to alter the contents of a read-only edit control. The boolean flag value determines whether the control will be left in read-only mode (true) or normal state (false).

SETLBTABS **routine to set tab stops in a list box control**

Syntax:

setlbtabs lbox, tablist%()

where:

lbox: a list control reference (name property value)
tablist:an array (passed by reference) of integers

Using this routine, a programmer can alter the tab stop position values in a list box that was created with the LB_HAS_TABSTOPS style. The standard visual basic listbox (not combo-box) supports tab stops. The array of integers holds the tab stop positions, calculated in *dialog box units*. A dialog box unit is equivalent of approximately one quarter of the average width of the system font. This awkward unit of measure is a byproduct of having proportional and sizeable fonts available.

It is important to note that the first array position will determine the position of the second column.

SET_3D **routine to set-up three-dimensional border line width and color**

Syntax:

set_3d white&, black&, lines
set_3d 0, 0, 3

where:

white: A long integer holding the windows RGB color for the "white" (non shadow) part of a three dimensional border.
black: A long integer holding the windows RGB color for the "black" (shadow) part of a three dimensional border.
lines: the number of one pixel width lines that make up the thickness of the border.

Windows RGB colors are three byte integers that contain a red, green, and blue component. The special values shown in the second syntax above (zero) indicates that the system defined colors for these shadow components should be determined and then used. System defined colors are set using the control panel application, color section, button highlight, button shadow values.

This routine controls the results of using the THREE_DEE routine described below

---

THREE_DEE **routine to draw a three dimensional shadow box around a control**

Syntax:

three_dee ctlname, mode%

where:

ctlname: a control reference (any control on any form)
mode: an integer (1, 0) indicating raised look (1), or sunken look (0)

Using this routine it is easy to create three dimensional controls, custom buttons, and artsy borders. It is suggested that the AUTOREDRAW property of a form using three_dee'd controls be set to true. Be careful to note that paint events no longer are generated if a forms AUTOREDRAW property is true. This should not impact design or function in any way, since PAINT is easily avoided.

TILECHILDREN **routine to rearrange children windows in tile mode on parent window**

Syntax:

tilechildren hwnd%, style%

where:

hwnd: A window handle of the parent window
style: (not used below windows version 3.1) Indicates vertical or horizontal tiling

Tiled windows take the child windows and arrange them to cover the parent as if they were ceramic tiles, laid out edge to edge. The style flag (boolean) impacts the arrangement. Windows performs the operations performed here, and the values of style can be determined in the windows SDK documentation.

UNDROP **routine to remove the drop-down list from a combo-box (hide list)**

Syntax:

undrop combobox

where:

combobox: combo box reference (name property value)

The only known use of this routine is to remove the list from view while processing change events in the combo box.

WNDCAPTION **function to return a window caption given a window handle**

Syntax:

s$ = windcaption(hwnd%)

where:

s: A string to hold the return value
hwnd: a window handle to locate and extract the caption from.

WRITEPROFILESTRING **function to store a value in the current application INI file**

Syntax:

i% = writeprofilestring(sect$, entry$, value$)

where:

i: An integer to hold the return value
sect: section name in INI file. e.g. [SAMPLE]
entry: item name within section. e.g. ITEM=somevalue
value: the value to place on the right side of the equal sign

This function stores the values passed in the current application INI file (see SETAPPNAME). These items may be later extracted with GETPROFILESTRING and GETPROFILEINT. If an entry does not yet exist when this function is called, the new entry/section will be created automatically.