TBAR.VBX

Toolbar Custom Control for Visual Basic 3.0

Contents

Introduction

TBAR is a custom control for implementing industry standard toolbars in Visual Basic. TBAR uses one bitmap, created by the developer, to display all buttons, in all button states. One bitmap means faster loading and fewer system resources. Additionally, you can design, define, and implement the behavior of your toolbar without writing a single line of code! No longer do you have to be envious of Visual C++ or write your own code to implement toolbars. TBAR will do everything for you. Here are some features:

- One bitmap defines all buttons, and all button states
- All standard button types allowed, push, toggle, disabled, enabled
- Group buttons. When one is toggled, the others become untoggled
- Pop up help for each button as seen in industry standard applications
- And more

ZIP File Contents

This ZIP file contains:

- TBAR.VBX
  VB version of the TBAR custom control
- **PROJECT1.MAK**
  Sample project file for TBAR
- **MDIFORM1.FRM**
  Sample MDI form with toolbar
- **MDIFORM1.FRX**
  File used by VB for MDIFORM1
- **FORM1.FRM**
  Form with some simple information
- **FORM2.FRM**
  Form with wrapping toolbar
- **READ.ME**
  This file
- **TOOLBAR.BMP**
  Bitmap of toolbar utilized by the sample

Registration

If you like and appreciate this custom control...

1) Small time developers like me, send $20.00 to the address listed above
2) Corporate users send $25.00 to the address listed above
3) Register in the CompuServe SWREG Forum ID 1700 for $15.00

Registered users may purchase the source...

Send $45.00 to me at the above address.
Register in the SWREG Id 2258.

This control is written in C++ and has been compiled using both Borland C++ 3.1 and Visual C++ 1.5.

You may now register this control in the SWREG on Compuserve. Type GO SWREG.

Corporate users should register via mail. Alternatively, you may register in SWREG by registering two copies, as many of you already have!

Warranty

The control, as is, works the way that I want it to. This does not mean that it will work the way that you want it to. Along those lines, I totally disclaim that the control will do anything whatsoever. This includes any implied abilities and any WARRANTIES, including those for SUITABILITY for a particular purpose, MERCHANTIBILITY, and all that other bull crap.

Call For Applications

I am being continuously asked to show applications which make use of my custom controls in publications featuring Visual Basic and VBXs. If any of you are interested in free advertising for your application, please drop me a line.

Reaching The Author

I can be reached via U.S. Mail at...

McKean Consulting
Robin W. McKean
1042 Braddock Circle
Woodstock GA 30188

You can reach me via E-Mail at the following locations...

CompuServe:

Robin W. McKean
72622,1403

Shareware South:

(404) 370-0872
Robin W. McKean
Atlanta, GA

I'm on CompuServe once or twice a day. I check Shareware South about once a week.

Revision History

1.00.0000	Initial release of the TBAR.VBX Custom Control
1.50.0000	Added properties for visible buttons, ids, bevels, autoplacement, rounded corners, and improved the look of disabled bitmaps.
1.51.0000	Fixed bug when loading a form saved as text
1.52.0000	Fixed button staying depressed when modal form is called up by button click event.
2.00.0000	Implemented popup help
2.10.0000	Various bug fixes. Added VERSION info.
2.11.0000	Remove balloon/popup help when top window no longer active

Acknowledgements

Thanks to James Moore for the bug report on Forms saved as text.

Thanks to Terje Enge for his instrumental help in beta testing TBAR 2.0.

Thanks to Kevin Klasman of Klasman Quality Consulting for converting README.TXT to Windows Help

Properties

The properties for this control can be broken down into two areas. The first are properties which apply to the whole toolbar; the second are properties which apply to individual buttons. Individual button properties will change as you change the CurrentButton property, or as you select a button on the screen, during design, with the right mouse button.

Toolbar Properties
Toolbar Button Properties

Toolbar Properties

AutoPlacement
BevelWidth
Bitmap
ButtonHeight
ButtonState
ButtonWidth
CurrentButton
EnableHelp
HelpBackColor
HelpForeColor
HelpInterval
HelpPosition
HelpQuickInterval
HelpStyle
ID
IDList
Interval
LeadingInterval
OutlineColor
Outline
PressedBevelWidth
RoundedCorners
ToolbarStyle
Version
VisibileButtons

Kevin M. Klasman
Klasman Quality Consulting
13 Vespa Lane
Nashua, NH 03060
Phone: (603) 886-3861 Compuserve: 70233,1476

Outline

This property determines whether or not their is a solid line separating the toolbar from its parent window. Most standard toolbars have a solid line of some sort to emphasize the toolbar. When the toolbar is aligned top, then there is a line at the bottom. When the toolbar is aligned bottom, there is a line at the top. When there is no alignment, than this property draws a border around the toolbar.

OutlineColor

This is the color of the toolbar outline. It's default is black.

Interval

This property represents the amount of space, in pixels, between the buttons in the toolbar. If you want all buttons to have a little space between them, set this property to 2. If you want more distinct button borders, set this property to one. The demo has this property set to 0.

ButtonWidth

This property tells TBAR how wide, in pixels, each button is in your BITMAP. TBAR will add 8 pixels to this value to determine the true button size. As you design your bitmap for your toolbar, you should decide beforehand what each button bitmap size will be. The demo bitmap is 16 pixels wide by 15 pixels high. There are ten buttons, so the bitmap is 160 pixels wide. Some resource editors allow you to break up your bitmap into grids, some don't. You could use PaintBrush to edit your bitmaps.

ButtonHeight

This property tells TBAR how high each bitmap is. This property is used, and may be used later to divide bitmaps horizontally as well as vertically. The bitmap provided with the demo is 15 pixels high.

Bitmap

This is the property for the bitmap which represents the buttons on your toolbar. There are some important things for you to remember when designing your bitmap. All important areas of your bitmap should be outlined in black. The black areas will be greyed when the bitmap is disabled. The normal background color, RGB(128,128,128) (LTGRAY), must only be used for the background. This color is masked out when re-drawing the button as pressed, toggled, and disabled. You may not use another background color. This is consistent with most toolbars currently in the industry. The demo uses black and white as the only colors. Microsoft (TM) uses these colors in all of their applications. You, however, may use other colors if it suits your purpose. The only restrictions are those mentioned above.

VisibileButtons

(Yea, I spelled it wrong, duh???)

This is the number of buttons visible on the screen. It is possible to change the number of visible buttons at run-time to un-hide buttons at the end of the toolbar. I must warn you. This feature is not tested. I have plans in the works to allow drag and drop of buttons on the toolbar. At that time, this feature will become more stable. For right now, it might be best to only set this property at design time. (But hey, try it if you want.)

LeadingInterval

This property is the number of pixels between the beginning of the toolbar and the first button on the toolbar.

CurrentButton

For setting button properties, this property determines which button will be affected. When you select a button with the right mouse button during design, this property will be set for you.

```
TBar1.CurrentButton = 0
TBar1.Gap = "5"
```

ButtonState

This property is not available at design time. You can change the enabled/disabled or toggled/untoggled state of a button by using this property. The following values are hard coded by the system:

```
BS_IS_TOGGLED                   1
BS_IS_ENABLED                   2
BS_IS_NOTTOGGLED                3
BS_IS_NOTENABLED                4
BS_IS_VISIBLE                   5
BS_IS_NOTVISIBLE                6
```

So, if I wanted to disable the second button (0 index) on the bitmap, I might do something like this:

```
TBar1.ButtonState(2) = 4
```

Since the states of the button are indeterminite, reading this property will not return anything meaningful. Additionally, in versions after 1.5, the index member of this property now corresponds to the ID of the bitmap, not the logical index of the bitmap. This allows the developer to change the button state of any button, no matter where it has been moved to by the user (if you are using AutoPlacement feature). These new enhancements are fully compatible with previous versions.

RoundedCorners

Setting this propery to True will cause the edges of all buttons to appear somewhat rounded.

ID

This property sets and gets the id of the the button at the requested index. This property allows you to set the bitmap of any button on the toolbar to any section of the bitmap your toolbar is using. For example:

Your bitmap is 100 wide by 50 deep. Each bitmap you are using for a button is 25 pixels wide (ButtonWidth) and 25 pixels high (ButtonHeight). So, the bitmap is divided into 4x2 bitmaps. To set the first button in your toolbar to the first bitmap on the second row, you can say:

```
TBar1.Id(0) = 4
```

Then, you can disable that same button by doing:

```
TBar1.ButtonState(4) = 4
' ButtonState uses Id for reference
```

You may use this property to get the ID(s) of each button in the toolbar as well.

AutoPlacement

Setting this property to True will allow the end user to drag and drop buttons on the toolbar to new places using the right mouse button. Try this one out and see if it fits into your application. You can save the placement using the IDList property and then restore the button order by setting this property when the application is started

IDList

This property returns a string of Ids for each button in the bitmap. You can then use this string later to restore the order of the buttons when the user restarts the program, or you can use it to set and show different sets of buttons in the toolbar.
For example:

```
' Restore previous button state
Dim A$ As String * 81
GetPrivateProfileString("Toolbar","ButtonOrder","0;1;2;3;4;5;6;7;8;9",A$,80,"
TOOLBAR.INI")
TBar1.IDList = A$
```

Now, say when the user pops up a certain document in an MDI application, you want to show four buttons which do a certain set of operations:

```
Command1_Click()
     TBar1.VisibileButtons = 4                ' Number of buttons on bar
     TBar1.IDList = "12;13;14;15"             ' Each buttons ID
End Sub
```

BevelWidth

This property controls the width of the white and dark grey lines around the outside of the button. Changing this property will enhance or reduce the 3D effects of the button. Valid bevel widths are between 1 and 10.

PressedBevelWidth

This property controls the width of the black lines around the outside of the button when it is pressed. Changing this property will enhance or reduce the 3D effects of the button. Valid pressed bevel widths are between 1 and 10. This value should always be less than or equal to the BevelWidth property.

Version

This property has been added to ensure backwards compatibility with all versions of TBAR.

HelpForeColor

This property controls the color of the text in your popup help window.

HelpBackColor

This property controls the color of the background in your popup help window.

EnableHelp

This flag determines whether or not help is enabled for your toolbar. Setting this property to True will cause the help window to be displayed when the user leaves the mouse over one spot on a button for HelpInterval number of milleseconds. See HelpText under Toolbar Button Properties for specifics on the help text.

HelpInterval

The delay in milleseconds before the help window is displayed. Timing starts when the user leaves the mouse cursor over a specific spot on the button.

HelpQuickInterval

This property is functionally equivelent to the HelpInterval property, except that this is the delay used after popup help has been shown for the first time, and the mouse has not left the toolbar.

HelpPosition

This property controls the location of the popup help window. The default is for the window to show up 22 pixels beneath the mouse cursor, aligned left. As an option, you may change this so that the popup help window will appear beneath the button.

HelpStyle

This feature is not implemented, but is reserved for future versions. What I need is a little help creating a ballon type help window. I don't know that much about trig. A little help from my users would be appreciated. The look I am looking for is a "cartoon" type balloon, with the help text in it.

ToolbarStyle

I added this property so that you can control the arranging of buttons on the toolbar. The following is a brief description of each option:

0 - Horizontal

The buttons are placed one after another horizontally.This is the default. The buttons will not wrap.

1 - Horizontal w/Wrap

The buttons are placed one after another horizontally. This is the default. The buttons will wrap if they run over the end of the toolbar. It is up to you to determine if the buttons are going to wrap, and resize your toolbar vertically. Note that any intervals will still apply, as will any gaps. This feature is mainly to be used to create a "toolbox" type toolbar.

2 - Vertical

The buttons are placed on after another vertically. The buttons will not wrap.

3 - Vertical w/Wrap

The buttons are placed on after another vertically. If the buttons would run over the end of the toolbar, the will wrap vertically. This is very similar to number 1. Either will work, I guess it just depends on what order you want the buttons to appear in your toolbar.

There is another custom control, **TBARWW**, available on CompuServe, which will allow you to make a form always stay on top, like a real toolbox. If you link that custom control with my toolbar, you will be able to create custom tool boxes.

Toolbar Button Properties

Toolbar button properties are stored as strings. So, you should always set these properties using strings.

ButtonVisible
Disabled
Gap
Group
HelpText
Sticky

Gap

This property represents the gap between CurrentButton and the next button. This property is stored in Pixels. The demo program uses this value to put spaces in between buttons in groups or which have similar purposes.

```
' Put 10 pixesl between CurrentButton and the next
TBar1.Gap = "10"
```

This property is mostly set at design time. But you can set it at run time.

Sticky

This property determines whether or not the button can be "toggled". Setting this property to "1" allows this button to remain pressed when the mouse is released, and then to be unpressed, when pressed with the mouse again. Buttons with this property set to "1" will respond to the Toggle event.

```
TBar1.CurrentButton = 1
TBar1.Sticky = "1"

TBar1.CurrentButton = 1
TBar1.Sticky = "0"                ' Button is not sticky
```

Disabled

This property determines whether or not the button can be pressed with the mouse. Setting this property to "1" disables all "Click" events for that button. Normal buttons on the toolbar provide a Click event when the mouse button is pressed then released on them. In addition, the button will appear as "greyed" on the screen.

```
TBar1.CurrentButton = 5
TBar1.Disabled = "1"

TBar1.CurrentButton = 5
TBar1.Disabled = "0"   ' Enable button
```

Group

This property should only be used by "Sticky" buttons. Sticky buttons can be grouped, so pressing one sticky button causes all other sticky buttons in the group to become unpressed. You can enter the index of any other sticky buttons in the group separated by a comma. This property should be set at design time, but you can read from and write to it at run time. It is a string property as well.

```
TBar1.CurrentButton = 8      ' Set the current button
TBar1.Group = "7,9"                ' Set the other buttons in his group
```

To disable the group set Group = "None"

```
Tbar1.CurrentButton = 3
TBar1.Group = "None"
```

ButtonVisible

This property defaults to True. Setting it to False causes that button to be invisible. Note that the button that is affected is the controlled by the CurrentButton property. Note additionally, that the CurrentButton property is a logical index into the button array. Along those lines, this property should really only be used at design time. Use the ButtonState property to hide and unhide buttons by ID during run time.

HelpText

This property controls what help text will be displayed when the help window is popped up. You may not include the semi-colon character (;) in your help text. If you want your help text divided up into more than one line, you can specify this by using the \r. For example:

TBar1.HelpText = "This text is divided into line one\rand line two"

Everything after the \r will appear on line two of the help window.

Events

Click
DoubleClick
MouseDown, MouseMove, MouseUp
Toggle

Click

This event is called when the user presses a "normal" button, and then releases the mouse over the button. The parameter "Button", passed to this event, tells the developer which button was pressed. Note that this parameter is now the button's ID, in case AutoPlacement is turned on, clicking the same button, no matter where it has been moved, will cause the same Click event to be generated.

```
TBar1_Click(Button As Integer)

     ' When save file is pressed, save the file, then disable the
     ' button until the current document is dirty again.
     Select Case Button
           Case 0:
                 NewDocument
           Case 1:
                 OpenDocument
           Case 2:
                 SaveToFile
                 TBar1.ButtonState(Button) = 4
     End Select
End Sub
```

Toggle

This event is called when the state of a "Sticky" button changes. If the sticky button is part of a group, then this event will be called twice, once for the button getting pressed, and once for the other button in the group which will be unpressed when the current button does down.

```
TBar1_Toggle(Button As Integer, State As Integer)

     Select Case Button
           Case 7:
                 SetAligment LeftJustify,State
           Case 8:
                 SetAlignment RightJustify,State
           Case 9:
                 SetAlignment CenterJustify,State
     End Select

End Sub
```

DoubleClick

This event is called when the user double clicks a button. I'm not sure what this will be used for, but it is provided in case a user has special needs.

MouseDown, MouseMove, MouseUp

These events are called when a mouse action has taken place upon the toolbar. Each one of these events take the following parameters:

```
TBar1_MouseDown(MouseButton As Integer, Shift As Integer, X As Single,
                            Y As Single, Button As Integer)
TBar1_MouseMove(MouseButton As Integer, Shift As Integer, X As Single,
                            Y As Single, Button As Integer)
TBar1_MouseUp(MouseButton As Integer, Shift As Integer, X As Single,
                            Y As Single, Button As Integer)
```

MouseButton indicates which mouse button caused the event. This parameter is equivalent to the Button parameter for normal MouseDown events, as in the VB Help File. This parameter is 1 for the left button, 2 for the right button, and 4 for the middle button.

Shift represents the state of the Ctrl and Shift keys. I did not include the Alt key. I didn't see the purpose for this, and the Alt key state is not provided as a default in the mouse event. I will include the Alt key state if anyone requests me to do so. The shift states are 1 for Shift down, and 2 for Control button down.

X is the location on the screen, in twips of the mouse events x location.

Y is the location on the screen, in twips of the mouse events y location.

Button is the toolbar button over which the current mouse event took place. If there was no button under the mouse event, then this parameter will be equal to -1.

New Features

Vertical and horizontal toolbars with wrapping