

Basic Constituents

Card Constituent

Button Constituent

Combo Box Constituent

Edit Constituent

Extended Combo Constituent

Header Constituent

Image List Constituent

List Constituent

List View Constituent

Panel Constituent

Progress Constituent

ReBar Constituent

Browser Constituent

Shell Constituent

Splitter Constituent

Status Bar Constituent

System Image List Constituent

Tab Constituent

Tip Constituent

Tool Bar Constituent

Tray Icon Constituent

Tree View Constituent


Zip Constituent

Card Constituent

Properties

Methods

Events

	Class: B3Card	Sample-Card\TestCard.vbp
---	----------------------	---------------------------------

Overview

The Card constituent allows you to design programs that use playing cards as components. The card contains the functionality to show all 52 cards in the deck and has 2 custom card backs. A card can easily be flipped over to show its back or face as needed.

Features

Using the combination of the Suit and Card properties, the control can be set to any of the 52 standard cards and the Joker. The card has 2 separate back styles (Red Diamond and Blue Diamond) which are set using the Back property. The following figure displays the Card sample program:

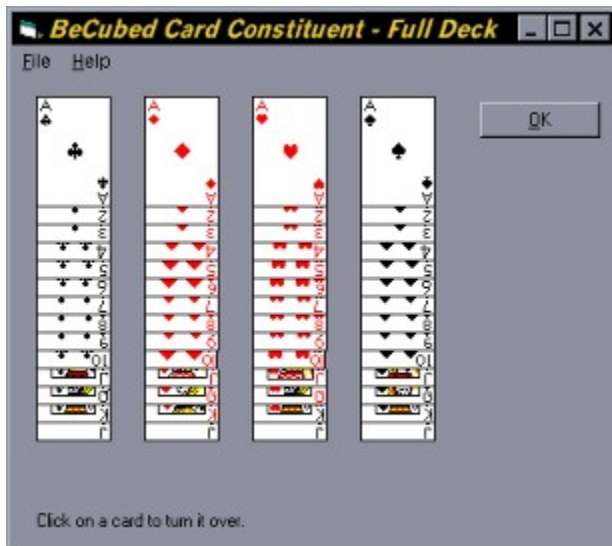


Figure 11: The BeCubed Card Constituent

In this sample, 4 Card constituents have been added to a form at design time (the Aces). At run time, the other cards are added as dynamic control array elements.

For example, to add the first column of cards (Clubs) the following code is used:

```
'clubs
For iCard = 0 To 13
    If iCard <> 0 Then Load B3CardClubs(iCard)
    StackCards B3CardClubs(iCard), BlueDiamond, Club, iCard + 1, lLastTop
    lLastTop = B3CardClubs(iCard).Top
Next iCard
```

When a card is clicked, it is turned over (opposite of the currently visible side). This is done as follows:

```
'if the card back is currently showing, flip it over.
```

```
If B3CardSpades(Index).Card = Back Then  
    B3CardSpades(Index).Card = Index + 1  
Else  
    'otherwise, turn it over.  
    B3CardSpades(Index).Card = 0  
End If
```

Properties (Card Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
Name	Standard Property	
AutoSize	Automatically sizes the control to fit the card bitmap.	
Back	Sets the card back to display. The choices are: 0 - BlueDiamond 1 - RedDiamond	
DragIcon	Standard Property	
DragMode	Standard Property	
Height	Standard Property	
Index	Standard Property	
Left	Standard Property	
Suit	Sets the card's suit. The choices are: 0 - Club 1 - Diamond 2 - Heart 3 - Spade	
Tag	Standard Property	
ToolTipText	Standard Property	
Top	Standard Property	
Visible	Standard Property	
WhatsThisHelpID	Standard Property	
Width	Standard Property	

Methods (Card Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Method Description	
AboutBox	Standard Method	
Drag	Standard Method	
Move	Standard Method	
ShowWhatsThis	Standard Method	
ZOrder	Standard Method	

Events (Card Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Event Description	
Click	Standard Event	
DoubleClick	Standard Event	
DragDrop	Standard Event	
DragOver	Standard Event	
MouseMove	Standard Event	
MouseDown	Standard Event	
MouseUp	Standard Event	

Button Constituent

Properties

Methods

Events



Class: bbbCMD

Sample-Button\CmdGrp.vbg

Overview

The BBBCMD constituent is what you might consider an all-in-one control. In Windows, the Button window class contains the functionality for the following controls:

- Command Button
- Option Button
- Check Box
- Group Button

There are several button state properties that the control provides that allow you to set the type of window you would like created (remember, all controls are actually windows). The following properties determine the type of control that is being created:

Property	Control
bsPUSHBUTTON	Command Button
bsRADIOBUTTON	Option Button
bsCHECKBOX	Check Box
bsGROUPBOX	Group Box

The bsGROUPBOX property allows the control to be used as a container to group other controls. The following section outlines the extended features of the BeCubed Button constituent control.

Features

This section covers the extended features of the Button control. This control is unique in that the button constituent can be used as 4 different controls. Since this is the case, each extended feature specifies which controls it applies to. For ease of use, this section only outlines features that are not provided by the standard controls.

Place the text to the left of a check box or option button.

The bsLEFTTEXT property places the text to the left of the check or option selector within the control. The bsRIGHTTEXT (default) places the text to the right of the selector.

Create an Owner Draw button (Command Button, Option Button, Check Box)

Owner-draw allows you to draw directly into a control. For example, you may want to draw an image to the left of an item in a list box. You can also use owner-draw to completely draw the control. This means that you are then responsible for managing all additional drawing, redrawing, etc.

The prototype for the drawing callback is as follows:

Sub DrawButton (objButton as Object, dw as DRAWITEMSTRUCT)

objButton is the control that is being painted; dw is a draw item structure.

The DrawButton procedure (DRAW.BAS) in the Button sample contains the source code necessary for completely drawing and managing a control.

Make an Option Button or Check Box act like a Command Button

Using the bsPUSHLIKE property, you can create an option button or check box that can be pushed like a command button. The button looks raised when it is not selected and sunken when it is.

Intercept Windows Messages with the Control

The WindowProcCB property is used to set the address of a function that is fired for every Windows message. This procedure is called before default processing has occurred. This allows the programmer to change the parameter values or cancel processing altogether. You can also use the WindowProcCBAfter property to set the procedure that is called after default processing has occurred. The same procedure can be used for both callbacks.

Properties (Button Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
BackColor	Standard Property	
bs3STATE	Creates a button that is the same as a check box, except that the box can be grayed as well as checked or unchecked. Use the grayed state to show that the state of the check box is not determined.	
bsAUTO3STATE	Creates a button that is the same as a three-state check box, except that the box changes its state when the user selects it. The state cycles through checked, grayed, and unchecked.	
bsAUTOCHECKBOX	Creates a button that is the same as a check box, except that the check state automatically toggles between checked and unchecked each time the user selects the check box.	
bsAUTORADIOBUTTON	Creates a button that is the same as a radio button, except that when the user selects it, Windows automatically sets the button's check state to checked and automatically sets the check state for all other buttons in the same group to unchecked.	
bsBITMAP	Specifies that the button displays a bitmap.	
bsBOTTOM	Places text at the bottom of the button rectangle.	
bsCENTER	Centers text horizontally in the button rectangle.	
bsCHECKBOX	Creates a small, empty check box with text. By default, the text is displayed to the right of the check box. To display the text to the left of the check box, combine this flag with the BS_LEFTTEXT style (or with the equivalent BS_RIGHTBUTTON style).	
bsDEFPUSHBUTTON	Creates a push button that behaves like a BS_PUSHBUTTON style button, but also has a heavy black border. If the button is in a dialog box, the user can select the button by pressing the ENTER key, even when the button does not have the input focus. This style is useful for enabling the user to quickly select the most likely (default) option.	
bsFLAT	Disables 3D effects.	
bsGROUPBOX	Creates a rectangle in which other controls can be grouped. Any text associated with this style is displayed	

in the rectangle's upper left corner.

bsICON	Specifies that the button displays an icon.
bsLEFT	Left-justifies the text in the button rectangle. However, if the button is a check box or radio button that does not have the BS_RIGHTBUTTON style, the text is left justified on the right side of the check box or radio button.
bsLEFTTEXT	Places text on the left side of the radio button or check box when combined with a radio button or check box style. Same as the BS_RIGHTBUTTON style.
bsMULTILINE	Wraps the button text to multiple lines if the text string is too long to fit on a single line in the button rectangle.
bsNOTIFY	Enables a button to send BN_DBLCLK, BN_KILLFOCUS, and BN_SETFOCUS notification messages to its parent window. Note that buttons send the BN_CLICKED notification message regardless of whether it has this style.
bsOWNERDRAW	Creates an owner-drawn button. The owner window receives a WM_MEASUREITEM message when the button is created and a WM_DRAWITEM message when a visual aspect of the button has changed. Do not combine the BS_OWNERDRAW style with any other button styles.
bsPUSHBUTTON	Creates a push button that posts a WM_COMMAND message to the owner window when the user selects the button.
bsPUSHLIKE	Makes a button (such as a check box, three-state check box, or radio button) look and act like a push button. The button looks raised when it isn't pushed or checked, and sunken when it is pushed or checked.
bsRADIOBUTTON	Creates a small circle with text. By default, the text is displayed to the right of the circle. To display the text to the left of the circle, combine this flag with the BS_LEFTTEXT style (or with the equivalent BS_RIGHTBUTTON style). Use radio buttons for groups of related, but mutually exclusive choices.
bsRIGHT	Right-justifies text in the button rectangle. However, if the button is a check box or radio button that does not have the BS_RIGHTBUTTON style, the text is right justified on the right side of the check box or radio button.
bsRIGHTBUTTON	Positions a radio button's circle or a check box's square on the right side of the button rectangle. Same as the

	BS_LEFTTEXT style.
bsTEXT	Specifies that the button displays text.
bsTOP	Places text at the top of the button rectangle.
bsUSERBUTTON	Obsolete, but provided for compatibility with 16-bit versions of Windows. Win32-based applications should use BS_OWNERDRAW instead.
bsVCENTER	Places text in the middle (vertically) of the button rectangle.
Caption	Returns or sets the display caption of the control
dwStyle	Holds the value of the combined bsValues from above. This property is for storage and should not be exposed or set directly.
Enabled	Standard Property
Font	Standard Property
ForeColor	Standard Property
hWnd	Standard Property
ItemDrawCB	Sets the address of the callback procedure when programming an owner draw button. This must be a procedure in a BSA module with a particular set of parameters. See the ButtonDraw documentation for details.
WindowProcCB	Sets the address of a callback procedure that will fire for every Windows message. This procedure will be called before default processing. The programmer can change values of parameters or can cancel the message processing altogether.
WindowProcCBAfter	Sets the address of a callback procedure that will fire for every Windows message. This procedure will be called after default processing. The same callback procedure can be used for both WindowProc callbacks.

Methods (Button Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>								
Method Name	Method Description									
AboutBox										
bmCLICK	<p>Simulates the user clicking a button. This message causes the button to receive a WM_LBUTTONDOWN and a WM_LBUTTONUP message, and the button's parent window to receive a BN_CLICKED notification message.</p> <p>object.bmCLICK()</p>									
bmGETCHECK	<p>Retrieves the check state of a radio button or check box.</p> <p>object.bmGETCHECK() As Integer</p> <p>Return Values: The return value from a button created with the BS_AUTOCHECKBOX, BS_AUTORADIOBUTTON, BS_AUTO3STATE, BS_CHECKBOX, BS_RADIOBUTTON, or BS_3STATE style can be one of the following:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>BST_CHECKED</td> <td>Button is checked.</td> </tr> <tr> <td>BST_INDETERMINATE</td> <td>Button is grayed, indicating an indeterminate state (applies only if the button has the BS_3STATE or BS_AUTO3STATE style).</td> </tr> <tr> <td>BST_UNCHECKED</td> <td>Button is unchecked</td> </tr> </tbody> </table> <p>If the button has any other style, the return value is zero.</p>		Value	Meaning	BST_CHECKED	Button is checked.	BST_INDETERMINATE	Button is grayed, indicating an indeterminate state (applies only if the button has the BS_3STATE or BS_AUTO3STATE style).	BST_UNCHECKED	Button is unchecked
Value	Meaning									
BST_CHECKED	Button is checked.									
BST_INDETERMINATE	Button is grayed, indicating an indeterminate state (applies only if the button has the BS_3STATE or BS_AUTO3STATE style).									
BST_UNCHECKED	Button is unchecked									
bmGETIMAGE	<p>Retrieves the handle of the image (icon or bitmap) associated with the button.</p> <p>object.bmGETIMAGE() As Long</p> <p>Return Values The return value is the handle of the image, if any; otherwise, it is NULL.</p>									
bmGETSTATE	<p>Determines the state of a button or check box.</p> <p>object.bmGETSTATE() As Integer</p> <p>Return Values The return value specifies the current state of the button. You can use the following bitmasks to extract information about the state:</p>									

Value	Meaning
BST_CHECKED	Indicates the button is checked.
BST_FOCUS	Specifies the focus state. A nonzero value indicates that the button has the keyboard focus.
BST_INDETERMINATE	Indicates the button is grayed because the state of the button is indeterminate. This value applies only if the button has the BS_3STATE or BS_AUTO3STATE style.
BST_PUSHED	Specifies the highlight state. A nonzero value indicates that the button is highlighted. A button is automatically highlighted when the user positions the cursor over it and presses and holds the left mouse button. The highlighting is removed when the user releases the mouse button.
BST_UNCHECKED	Indicates the button is unchecked. Same as the Windows NT return value of zero.

bmSETCHECK Sets the check state of a radio button or check box.

object.bmSETCHECK(fCheck As Integer)

Parameters

fCheck

Specifies the check state. This parameter can be one of the following values:

Value	Meaning
BST_CHECKED	Sets the button state to checked.
BST_INDETERMINATE	Sets the button state to grayed, indicating an indeterminate state. Use this value only if the button has the BS_3STATE or BS_AUTO3STATE style.
BST_UNCHECKED	Sets the button state to unchecked

bmSETIMAGE Associates a new image (icon or bitmap) with the button.

object.bmSETIMAGE(image As Long)

wParam = (WPARAM) fImageType; // image-type
flag

lParam = (LPARAM) (HANDLE) hImage; // handle of
the image

Parameters

hImage

Identifies the image to associate with the button.

Return Values

The return value is the handle of the image previously
associated with the button, if any; otherwise, it is 0.

DoClick

Simulates a mouse click on the control. Returns nothing.

Refresh

Standard Method

Events (Button Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Event Description	
Click	Standard Event	
Command	The equivalent of the Windows WM_COMMAND message. This duplicates some of the functionality found in the Click and a few other messages, but this may be more convenient or more flexible to use. Command (wp as Long, lp as Long)	
DbClick	Standard Event	
KeyDown	Standard Event	
KeyPress	Standard Event	
KeyUp	Standard Event	
MouseDown	Standard Event	
MouseMove	Standard Event	
MouseUp	Standard Event	

CALL BACKS

Callback Name	Callback Description
DrawButton	<p>DrawButton is a prototype for a call back procedure. It does not have to be named DrawButton, but it must have two parameters with the correct type and be a SUB (not a function returning a value).</p> <p>You place the address of this routine into the ItemDrawCB property and set the bsOWNERDRAW property to True and this routine will be called to perform the painting of the button.</p> <p>Sub DrawButton (mybut As Object, dw As DRAWITEMSTRUCT)</p> <p>mybut the control being painted. You can access the value of the properties from this Object.</p> <p>dw is a draw item structure as used in all owner draw controls in Windows.</p>
WinProc	<p>A callback procedure used to respond to or process any Windows message. The button control is already subclassed for you eliminating the need to write or use a subclassing control to get to any Windows' message.</p>

Sub WinProc(mybut As Object, msg As Long, wp As Long, lp As Long, _ bDone As Long)

mybut the control this message belongs to. You can access the value of the properties from this Object

msg the message value, WM_...

wp the WPARAM value for the message.

lp the LPARAM value for the message.

bDone a flag that can be set to skip processing this message, also indicates before or after processing on entry.

Notice that the WinProc contains essentially the same information as any Windows message. The typical hWnd parameter has been replaced with the control and the bDone parameter has been added. It is possible and may be necessary to do some creative programming to get the value that is represented by the lp parameter as this value could be a direct long value or a pointer to almost any type of data. bDone is set to FALSE on entry to the WinProc Sub when firing the event for before default processing. It will be TRUE when the routine is processing the message after default processing. When bDone is FALSE, it may be set to TRUE before exiting the routine to skip default processing. When this is done, the “after” event will not be fired.

NOTE: The callback procedures supplied for this control can be very tricky to implement. There are several places in the Basic ActiveX control where the Address of the procedures used must be set into the variables and one place where they must be cleared. The DrawButton callback will never be fired (even if ItemDrawCB is set), unless the style of the button is set to owner draw. If you use the owner draw style for a button, you are not only responsible for the entire rendering of the button, but most of the functionality as well.

If you choose to use the “subclass” callbacks, remember that these callbacks carry a larger overhead due to the fact that the callback routine will be called for every message the button receives. Be sure not to set these values if you are not really using the WinProc Sub.

Combo Box Constituent

Properties

Methods

Events



Class: hhhCombo

Sample-Combo\hCombo.vbg

Overview

Combo boxes can be characterized by type and style. Combo box types determine whether the control's list is a drop-down list, and whether the selection field is an edit control. A drop-down list appears only when the user opens it so it uses less screen space than a list that is always visible. If the selection field is an edit control, the user can enter information that is not already contained in the list. Otherwise, the user may only choose from the items in the list.

The following table shows the three combo box types and indicates whether each includes a drop-down list and an edit control:

Combo box type	Drop-down list	Edit control
Drop-down combo box	Yes	Yes
Drop-down list box	Yes	No
Simple combo box	No	Yes

Combo box styles define specific properties of a combo box. An application can combine styles; however some styles only apply to certain combo box types. The cbsXXXXX properties, listed in the reference section later in this chapter, describe the various styles. The next section outlines some of the custom features of the Combo Box constituent.

Features

Use the control as a file list

The cbDIR method can be used to add the files from a specific directory to the combo box. This method has 2 parameters. The first contains the attributes of the files to select. This attributes parameter allows you to specify anything from a list of drives on the computer, to a list of Directories for a drive, to a list of files for a directory. You can also specify whether the list should include hidden, system, read only files, etc. The syntax for this method is as follows:

```
lRet = BBBCombo1.cbDIR (lAttributes As Long, sDir As String)
```

See the reference section for more information on this method.

Search for a specific item in the combo box

The cbFINDSTRING and cbFINDSTRINGEXACT methods are used to search string in the combo box. CbFINDSTRING finds the first item that begins with a specific string and cbFINDSTRINGEXACT searches for an exact match. These methods take 2 parameters: The index to begin searching at (-1 begins searching at the beginning of the list), and the string to search for. The following line of code searches for an exact match on the string "Hello", and starts with the first item in the combo box:

```
lIndex = -1  
sString = "HELLO"  
BBBComboBox1.cbFINDSTRINGEXACT(lIndex, sString)
```

Get the coordinates of all items in the drop-down list

The cbGETDROPPEDCONTROLRECT method returns the bounding rectangle of the dropped down list of items in the combo box. The method takes one parameter, a rectangle structure that is filled with the coordinates.

Determine if the drop-down list is open

In order to determine whether the drop-down list is open or closed, use the cbGETDROPPEDSTATE method. This

method takes no parameters and returns True if the list is open:

```
Dim bRet As Boolean  
bRet = BBBCombo1.cbGETDROPPEDSTATE ()
```

Get the width of the drop down list

The cbGETDROPPEDWIDTH method looks through the items in the combo box and returns the minimum allowable width of the drop down list.

Retrieve the height of an item in the Combo Box

The cbGETITEMHEIGHT method can be used to retrieve the height of an item. It takes the index of the item to retrieve the height for as a parameter:

```
lHeight = BBBComboBox1.CbGETITEMHEIGHT (1)
```

Retrieve the index of the first visible item in the Combo Box

The cbGETTOPINDEX method is used to retrieve the first visible item.

Programmatically select text in a drop-down combo box

The cbEDITSEL method takes the starting and ending points for text selection. The following line of code selects the first 5 character of an item:

```
BBBComboBox1.cbSETEDITSEL (1, 5)
```

Properties (Combo Box Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
BackColor	Standard Property	
BorderStyle	Standard Property	
cbsAUTOHSCROLL	Automatically scrolls the text in an edit control to the right when the user types a character at the end of the line. If this style is not set, only text that fits within the rectangular boundary is allowed.	
cbsDISABLENOSCROLL	Shows a disabled vertical scroll bar in the list box when the box does not contain enough items to scroll. Without this style, the scroll bar is hidden when the list box does not contain enough items.	
cbsDROPDOWN	Specifies a drop-down combo box.	
cbsDROPDOWNLIST	Specifies a drop-down list box.	
cbsHASSTRINGS	Specifies that an owner-drawn combo box contains items consisting of strings. The combo box maintains the memory and address for the strings so the application can use the <u>cbGETLBTEXT</u> message to retrieve the text for a particular item.	
cbsLOWERCASE	Converts to lowercase all text in both the selection field and the list.	
cbsNOINTEGRAL_HEIGHT	Specifies that the size of the combo box is exactly the size specified by the application when it created the combo box. Normally, Windows sizes a combo box so that it does not display partial items.	
cbsOEMCONVERT	Converts text entered in the combo box edit control from the Windows character set to the OEM character set and then back to the Windows set. This ensures proper character conversion when the application calls the <u>CharToOem</u> function to convert a Windows string in the combo box to OEM characters. This style is most useful for combo boxes that contain filenames and applies only to combo boxes created with the CBS_SIMPLE or CBS_DROPDOWN style.	
cbsOWNERDRAW_FIXED	Specifies that the owner of the list box is responsible for drawing its contents and that the items in the list box are all the same height. The owner window receives a <u>WM_MEASUREITEM</u> message when the combo box is created and a <u>WM_DRAWITEM</u> message when a visual aspect of the combo box has changed.	
cbsOWNERDRAW_VARIABLE	Specifies that the owner of the list box is responsible for drawing its contents and that the items in the list box are variable in height. The owner window receives a <u>WM_MEASUREITEM</u> message for each item in the combo box when you create the combo box and a <u>WM_DRAWITEM</u> message when a visual aspect of the combo box has changed.	
cbsSIMPLE	Specifies a simple combo box.	
cbsSORT	Automatically sorts strings added to the list box.	
cbsUPPERCASE	Converts to uppercase all text in both the selection field	

	and the list.
Container	Standard Property
CtlHeight	
DragIcon	Standard Property
DragMode	Standard Property
dwStyle	Holds the value of the combined bsValues from above. This property is for storage and should not be exposed or set directly.
Enabled	Standard Property
Font	Standard Property
ForeColor	Standard Property
Height, Width	Standard Property
HelpContextID	Standard Property
hWnd	Standard Property
Index	Standard Property
ItemCompareCB	
ItemData	
ItemDrawCB	Sets the address of the callback procedure for drawing when programming an owner draw list. This must be a procedure in a BASIC module with a particular set of parameters. See the ItemDraw documentation for details.
ItemMeasureCB	Sets the address of the callback procedure for measuring items when programming an owner draw list. This must be a procedure in a BASIC module with a particular set of parameters. See the ItemDraw documentation for details.
Left, Top	Standard Property
List	Standard Property
ListCount	Standard Property
ListIndex	Standard Property
MousePointer	Standard Property
Object	Standard Property
Parent	Standard Property
TabIndex	Standard Property
TabStop	Standard Property
Tag	Standard Property
Text	Standard Property
ToolTipText	Standard Property
TopIndex	Standard Property
Visible	Standard Property
WhatsThisHelpID	Standard Property
WindowProcCB	Sets the address of a callback procedure that will fire for every Windows message. This procedure will be called before default processing. The programmer can change values of parameters or can cancel the message processing altogether.
WindowProcCBAfter	Sets the address of a callback procedure that will fire for every Windows message. This procedure will be called after default processing. The same callback procedure can be used for both WindowProc callbacks.

Methods (Combo Box Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Method Description	
AboutBox		
AddItem	Standard Method	
cbADDSTRING	adds a list item.	
	object.cbADDSTRING(sString As String) As Long	
	Parameters sString - specifies the string to add to the combo box.	
	Returns The index of the item added.	
cbDELETESSTRING	deletes a list item.	
	object.cbDELETESSTRING(lIndex As Long) As Long	
	Parameters lIndex - item to delete.	
	Returns the number of items left in the list.	
cbDIR	Adds the filenames matching the specified attributes and path to the list.	
	object.cbDIR(lAttributes As Long, sDir As String) As Long	
	Parameters lAttributes - Specifies the attributes of the files to be added to the list box. This parameter can be a combination of the following values:	
	Value	Description
	DDL_ARCHIVE	Includes archived files.
	DDL_DIRECTORY	Includes subdirectories. Subdirectory names are enclosed in square brackets ([]).
	DDL_DRIVES	Includes drives. Drives are listed in the form [-x-], where x is the drive letter.
	DDL_EXCLUSIVE	Includes only files with the specified attributes. By default, read-write files are listed even if DDL_READWRITE is not specified.
	DDL_HIDDEN	Includes hidden files.

DDL_READONLY Includes read-only files.
DDL_READWRITE Includes read-write files with
no additional attributes.
DDL_SYSTEM Includes system files.

sDir - specifies the filename to add to the list. If the filename contains wildcards (for example, *.*), all files that match the wildcards and have the attributes specified by the lAttributes parameter are added to the list.

Returns
The 0 based index of the last file added to the list.

cbFINDSTRING returns the index of the first list item that begins with the specified text.

object.cbFINDSTRING(lIndexStart As Long, sString As String) As Long

Parameters
lIndexStart - specifies the index to begin the search, -1 starts at the beginning.

sString - specifies the string to search for.

Returns
The index of the item matching the string or -1 if none were found.

cbFINDSTRINGEXACT returns the index of the first list item exactly matching the specified text.

object.cbFINDSTRINGEXACT(lIndexStart As Long, sString As String) As Long

Parameters
lIndexStart - specifies the index to begin the search, -1 starts at the beginning.

sString - specifies the string to search for.

Returns
The index of the item matching the string or -1 if none were found.

cbGETCOUNT returns the number of list items.

object.cbGETCOUNT() As Long

Returns
the count of items in the list.

cbGETCURSEL returns the index of the currently selected item, if any.

object.cbGETCURSEL() As Long

Returns
the index of the currently selected item or -1 if none are selected.

cbGETDROPPED_
CONTROLRECT

Fills the specified rectangle structure with the screen coordinates of a drop-down list.

object.cbGETDROPPEDCONTROLRECT(rect As Long)

Parameters
rect - a long representing the first long in a rect structure. This will fill the rest of the rect structure with the values.

cbGETDROPPEDSTATE

Returns TRUE if a drop-down list is open; otherwise, it returns FALSE.

object.cbGETDROPPEDSTATE() As Boolean

Returns
True if open, otherwise False.

cbGETDROPPED_
WIDTH

Returns the minimum allowable width, in pixels, of the drop down list.

object.cbGETDROPPEDWIDTH() As Long

Returns
the minimum width of the dropped list box.

cbGETEDITSEL

returns the starting and ending position of the current selection. In drop-down list boxes, the window procedure returns an error.

object.cbGETEDITSEL(lStart As Long, lStop As Long)
As Long

Parameters
lStart = carat position within the edit control to begin the selection of text.

lStop = carat position within the edit control to end the selection of text.

Returns
zero-based 32-bit value with the starting position of the selection in the low-order word and with the ending position of the first character after the last selected character in the high-order word.

cbGETEXTENDEDUI

Returns TRUE if the combo box is a drop-down combo box or drop-down list box and the extend user-interface flag is set; otherwise, it returns FALSE.

object.cbGETEXTENDEDUI() As Boolean

Returns
 True if using the extended UI, otherwise False. By default, the F4 key opens or closes the list and the DOWN ARROW changes the current selection. In a combo box with the extended user interface, the F4 key is disabled and pressing the DOWN ARROW key opens the drop-down list.

cbGETHORIZONTAL_EXTENT returns the scrollable width, in pixels, of the drop down list.
 object.cbGETHORIZONTAL_EXTENT() As Long
 Returns
 the scrollable width, in pixels.

cbGETITEMDATA returns the 32-bit value associated with the specified list item.
 object.cbGETITEMDATA(IIndex As Long) As Long
 Parameters
 IIndex - specifies the item.
 Returns
 the 32 bit "item data" value. This value is set through cbSETITEMDATA.

cbGETITEMHEIGHT returns the height, in pixels, of the specified owner-drawn list item.
 object.cbGETITEMHEIGHT(IIndex As Long) As Long
 Parameters
 IIndex - this parameter must be -1 to retrieve the height of the selection field. It must be zero to retrieve the height of list items, unless the combo box has the cbsOWNERDRAWVARIABLE style. In that case, the index parameter is the zero-based index of a specific list item.
 Returns
 the height of the specified item.

cbGETLBTEXT copies the specified list text to the specified buffer.
 object.cbGETLBTEXT(IIndex As Long, sString As String) As Long
 Parameters
 IIndex - specifies the item from which to retrieve the text.
 sString - specifies the string to place the text in, this string must be initialized to at least the length of the text + 1.
 Returns
 the length of the text of the item.

cbGETLBTEXTLEN returns the length, in bytes, of the specified list text.
 object.cbGETLBTEXTLEN(IIndex As Long) As Long

	<p>Parameters</p> <p>lIndex - specifies the item from which to retrieve the text length.</p>
cbGETLOCALE	<p>Returns</p> <p>the length of the text of the item.</p> <p>returns the current locale for the list.</p>
cbGETTOPINDEX	<p>object.cbGETLOCALE() As Long</p> <p>returns the index of the first visible item in the drop down list.</p>
cbINITSTORAGE	<p>object.cbGETTOPINDEX() As Long</p> <p>initializes space for the specified number of items and the specified number of bytes for item strings.</p> <p>object.cbINITSTORAGE(lItems As Long, lMem As Long) As Long</p>
cbINSERTSTRING	<p>Parameters</p> <p>lItems - specifies the number of items to add</p> <p>lMem - specifies the amount of memory to allocate for strings.</p> <p>Returns</p> <p>maximum number of items that the memory object can store.</p> <p>inserts a list item at the specified position.</p> <p>object.cbINSERTSTRING(lIndex As Long, sString As String) As Long</p>
cbLIMITTEXT	<p>Parameters</p> <p>lIndex - the 0 based index that indicates the point at which to insert the string. If -1 is specified, the string is inserted at the end of the list.</p> <p>sString - specifies the string to insert.</p> <p>Returns</p> <p>the index at the point the string was inserted.</p> <p>sets the maximum number of characters a user can enter in the edit control. In drop-down list boxes, the window procedure returns an error.</p> <p>object.cbLIMITTEXT(lLimit As Long)</p>
cbRESETCONTENT	<p>Parameters</p> <p>lLimit - specifies the maximum number of characters that can be typed into the edit portion of a combo box.</p> <p>removes the contents of the list.</p> <p>object.cbRESETCONTENT()</p>

cbSELECTSTRING selects the first list item, if any, that begins with the characters in the specified text.

object.cbSELECTSTRING(IIndex As Long, sString As String) As Long

Parameters
IIndex - indicates the position within the list to begin the search. Setting this -1 will begin the search at the beginning of the list.

sString - specifies the prefix string to search for.

Returns
The index of the item selected or -1 if the item was not found.

cbSETCURSEL sets the current selection.

object.cbSETCURSEL(IIndex As Long) As Long

Parameters
IIndex - index of the item to select or -1 to deselect all items.

Returns
the item selected or -1 if none selected.

cbSETDROPPEDWIDTH Sets the minimum allowable width, in pixels, of the drop down list.
H

object.cbSETDROPPEDWIDTH(IWidth As Long) As Long

Parameters
IWidth - specifies the minimum width allowable for the drop list.

Returns
the new width of the list box or -1 if it fails.

cbSETEDITSEL selects the specified range of text. In drop-down list boxes, the window procedure returns an error.

object.cbSETEDITSEL(IStart As Long, IStop As Long) As Long

Parameters
IStart - specifies the starting position of selected text.

ISTop - specifies the ending position of selected text.

Returns
If successful 1 is returned, if it fails -1 is returned.

cbSETEXTENDEDUI Sets or clears the extended user-interface flag. This flag

changes the keys that open and close the list in a drop-down combo box or drop-down list box. If the combo box is a simple combo box, the window procedure returns an error.

object.cbSETEXTENDEDUI(bUI As Boolean) As Boolean

Parameters

bUI - if True, turns on the extended UI, if False turn it off.

Returns

Success or failure.

cbSETHORIZONTAL_EXTENT sets the scrollable width, in pixels, of the drop down list.

object.cbSETHORIZONTAL_EXTENT(IWidth As Long)

Parameters

IWidth - specifies the drop down list width.

cbSETITEMDATA associates the specified 32-bit value with a list item.

object.cbSETITEMDATA(IIndex As Long, IData As Long) As Boolean

Parameters

IIndex - specifies the item to associate the IData value with.

IData - the value to associate with the specified ite.

Returns

Success or failure.

cbSETITEMHEIGHT sets the height of the specified owner-drawn list item or the selection field.

object.cbSETITEMHEIGHT(IIndex As Long, IHeight As Long) As Boolean

Parameters

IIndex - specifies the item.

IHeight - specifies the height

Returns

Success or failure

cbSETLOCALE sets the current locale for the list. The locale affects how the list is sorted if it has the cbsSORT style and strings are added using cbADDSTRING.

object.cbSETLOCALE(ILocale As Long) As Long

Parameters

ILocale - specifies the new locale

Returns

the old locale.

cbSETTOPINDEX scrolls the drop down list so the specified item is at the top of the visible range.

object.cbSETTOPINDEX(IIndex As Long) As Boolean

Parameters

IIndex - specifies the 0 based item.

Returns

Success or Failure

cbSHOWDROPDOWN Shows or hides the drop-down list. This message has no effect on simple combo boxes.

object.cbSHOWDROPDOWN(bShow As Boolean)

Parameters

bShow - if True, shows the drop down list, if False hides the drop down list.

Clear	Standard Method
Drag	Standard Method
FontSelect	
Move	Standard Method
Refresh	Standard Method
ReleaseFont	
RemoveItem	Standard Method
SetFocus	Standard Method
ShowWhatsThis	Standard Method

Events (Combo Box Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Event Description	
Click	Standard Event	
Command	Processes notification messages from the edit control and list window and sends corresponding combo box notification messages to the parent window. For edit control notifications, the window procedure may update the list window's current selection, caret index, and top index. For list notification messages, the window procedure may update the content of the selection field.	
	CBN_CLOSEUP	Indicates the list in a drop-down combo box or drop-down list box is about to close.
	CBN_DBLCLK	Indicates the user has double-clicked a list item in a simple combo box.
	CBN_DROPDOWN	Indicates the list in a drop-down combo box or drop-down list box is about to open.
	CBN_EDITCHANGE	Indicates the user has changed the text in the edit control of a simple or drop-down combo box. This notification message is sent after the altered text is displayed.
	CBN_EDITUPDATE	Indicates the user has changed the text in the edit control of a simple or drop-down combo box. This notification message is sent before the altered text is displayed.
	CBN_ERRSPACE	Indicates the combo box cannot allocate enough memory to carry out a request, such as adding a list item.
	CBN_KILLFOCUS	Indicates the combo box is about to lose the input focus.
	CBN_SELCHANGE	Indicates the current selection has changed.
	CBN_SELENDCANCEL	Indicates that the selection made in the drop down list, while it was dropped down, should be ignored.
	CBN_SELENDOK	Indicates that the selection made drop down list, while it

	CBN_SETFOCUS	was dropped down, should be accepted. Indicates the combo box has received the input focus.
DbClick	Standard Event	
DragDrop	Standard Event	
DragOver	Standard Event	
GotFocus	Standard Event	
KeyDown	Standard Event	
KeyPress	Standard Event	
KeyUp	Standard Event	
LostFocus	Standard Event	
MouseDown	Standard Event	
MouseMove	Standard Event	
MouseUp	Standard Event	
Resize	Standard Event	

CALL BACKS

Callback Name

Callback Description

DrawItem

DrawItem is a prototype for a call back procedure. It does not have to be named DrawItem, but it must have two parameters with the correct type and be a SUB (not a function returning a value).

You place the address of this routine into the ItemDrawCB property and set the cbOWNERDRAW property to True and this routine will be called to perform the painting of line items.

Sub DrawItem (objItem As Object, dw As DRAWITEMSTRUCT)

objItem the control being painted. You can access the value of the properties from this Object.

dw is a draw item structure as used in all owner draw controls in Windows.

MeasureItem

MeasureItem is a prototype for a call back procedure. It does not have to be named MeasureItem, but it must have two parameters with the correct type and be a SUB (not a function returning a value).

You place the address of this routine into the ItemMeasureCB property and set the cbOWNERDRAW property to True and this routine will be

called to perform the sizing of an owner draw item.

Sub MeasureItem (objItem As Object, dw As
MEASUREITEMSTRUCT)

objItem the control being painted. You can access the value of the properties from this Object.

dw is a draw item structure as used in all owner draw controls in Windows.

WindowProc

A callback procedure used to respond to or process any Windows message. The combo box control is already subclassed for you eliminating the need to write or use a subclassing control to get to any Windows' message.

Sub WinProc(mybut As Object, msg As Long, wp As Long, lp As
Long, _ bDone As Long)

mycom the control this message belongs to. You can access the value of the properties from this Object

msg the message value, WM_...

wp the WPARAM value for the message.

lp the LPARAM value for the message.

bDone a flag that can be set to skip processing this message, also indicates before or after processing on entry.

Notice that the WinProc contains essentially the same information as any Windows message. The typical hWnd parameter has been replaced with the control and the bDone parameter has been added. It is possible and may be necessary to do some creative programming to get the value that is represented by the lp parameter as this value could be a direct long value or a pointer to almost any type of data. bDone is set to FALSE on entry to the WinProc Sub when firing the event for before default processing. It will be TRUE when the routine is processing the message after default processing. When bDone is FALSE, it may be set to TRUE before exiting the routine to skip default processing. When this is done, the "after" event will not be fired.

Edit Constituent

Properties

Methods

Events

BI Class: `bbbEdit Sample-Edit\Edit.vbg`

Overview

The Visual Basic equivalent of the edit control is the text box. Edit is the base window class used to create this control. The Basic Constituents edit control is an extended version of the Visual Basic text box. The control contains features that allow you to set the text in the edit constituent to read-only, capitalize the text in the control as it is entered, paste new text into the current selection and much more. The following section outlines some of the features of the edit constituent.

Features

Set the text in the edit constituent to read-only

To set the text to read-only, use the `esREADONLY` property. This prevents the user from modifying text displayed in an edit control.

Capitalize all characters displayed in the edit control

Characters in the edit control can be capitalized by using the `esUPPERCASE` property. This converts the characters as they are typed into the edit control.

Undo the entry of text into the edit control

If you want to undo an entry that a user has made, use the `emUNDO` method. This method takes no parameters and returns a boolean indicating whether or not the operation was successful.

Get a count of lines in a multiline edit control

If an edit control is set to multiline, the text wraps when the end of the line is reached. In order to determine the number of lines of text displayed in the edit control, use the `emGETLINECOUNT` method. This method takes no parameters and returns the number of lines of text in a multiline edit control. If no text is in the edit control, the return value is 1.

Determine whether the content of an edit control has been modified

To determine if the contents of an edit control has been modified, use the `emGETMODIFY` method. This method takes no parameters and returns a boolean indicating if the text has been modified.

Retrieve the position of the scrollbar thumb in a multiline edit control

To determine the position of the scrollbar thumb in an edit control, use the `emGETTHUMB` method. This method takes no parameters and returns the position of the scrollbar.

Hide the selection in an edit control

To hide the current selection in the edit control, use the `emHIDESELECTION` method. See the control reference for more information about the parameters to this method.

Scroll text horizontally or vertically

The `emLINESCROLL` method allows you to scroll the text in the edit control either horizontally or vertically. This method takes 2 parameters: The number of characters to scroll horizontally and vertically respectively. The method returns `True` if successful.

Paste new text over the current selection

To paste new text over the current selection, use the `emREPLACESSEL` method. This method takes a parameter indicating whether or not the replacement can be undone along with the new text.

Properties (Edit Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
Appearance		
BackColor		
BorderStyle		
Enabled		
esAUTOHSCROLL	Automatically scrolls text to the right when the user types a character at the end of the line. When the user presses the <ENTER> key, the control scrolls all text back to position 0.	
esAUTOVSCROLL	Automatically scrolls text up one page when the user presses <ENTER> on the last line.	
esCENTER	Centers text in a multiline edit control.	
esLEFT	Aligns text flush left.	
esLOWERCASE	Converts all characters to lowercase as they are typed into the edit control.	
esMULTILINE	Designates a multiple-line edit control.	
esNOHIDSEL	Normally, an edit control hides the selection when the control loses the input focus and inverts the selection when the control receives the input focus. Setting this property to True leaves the selection inverted all of the time.	
esOEMCONVERT	Text entered in the edit control is converted from the ANSI character set to the OEM character set and then back to ANSI. This style is most useful for edit controls that contain filenames.	
esPASSWORD	Displays all characters as an asterisk (*) as they are typed into the edit control. An application can use the SetPasswordChar member function to change the character that is displayed.	
esREADONLY	Prevents the user from entering or editing text in the edit control.	
esRIGHT	Aligns text flush right in a multiline edit control.	
esUPPERCASE	Converts all characters to uppercase as they are typed into the edit control.	

esWANTRETURN

Specifies that a carriage return be inserted when the user presses the ENTER key while entering text into a multiple-line edit control in a dialog box. Without this style, pressing the ENTER key has the same effect as pressing the dialog box's default pushbutton. This style has no effect on a single-line edit control.

Font

ForeColor

hWnd

MaxLength

ReadyState

Text

wsHSCROLL

wsVSCROLL

Methods (Edit Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Method Description	
AboutBox		
emCANPASTE	<p>Determines whether an edit control can paste a specified clipboard format.</p> <p>object.emCANPASTE(uFormat As Integer) As Boolean</p> <p>Parameters</p> <p>uFormat Value identifying the clipboard format to try, or zero to try any format currently on the clipboard.</p> <p>Return Values</p> <p>Returns a nonzero value if the clipboard format can be pasted or zero otherwise.</p>	
emCANUNDO	<p>Determines whether an edit-control operation can be undone; that is, whether the control can respond to the emUNDO method.</p> <p>object.emCANUNDO() As Boolean</p> <p>Return Values</p> <p>If the edit control can correctly process the EM_UNDO message, the return value is TRUE; otherwise, it is FALSE.</p>	
emCHARFROMPOS	<p>Retrieves the zero-based character index and zero-based line index of the character nearest the specified point in an edit control.</p> <p>object.emCHARFROMPOS(xCoord As Integer, yCoord As Integer, CharIdx As Integer, LineIdx As Integer)</p> <p>Parameters</p> <p>xCoord Specifies the x-coordinate of a point in the edit control's client area. The coordinate is relative to the upper-left corner of the client area.</p> <p>yCoord Specifies the y-coordinate of a point in the edit control's client area. The coordinate is relative to the upper-left corner of the client area.</p> <p>CharIdx Returns the 0 based character index for the character.</p>	

	<p>CharIdy Returns the 0 based line index for the character.</p>
emEMPTYUNDOBUFFER	<p>Resets the undo flag of an edit control. The undo flag is set whenever an operation within the edit control can be undone.</p> <p>object.emEMPTYUNDOBUFFER()</p> <p>Remarks The undo flag is automatically reset whenever the edit control receives a WM_SETTEXT or EM_SETHANDLE message.</p>
emFMTLINES	<p>Sets the inclusion flag of soft linebreak characters on or off within a multiline edit control. A soft linebreak consists of two carriage returns and a linefeed and is inserted at the end of a line that is broken because of wordwrapping.</p> <p>object.emFMTLINES(fAddEOL As Boolean) As Boolean</p> <p>Parameters fAddEOL Specifies whether soft-linebreak characters are to be inserted. A value of TRUE inserts the characters; a value of FALSE removes them.</p> <p>Return Values The return value is identical to the fAddEOL parameter.</p>
emGETFIRSTVISIBLELINE	<p>Determines the uppermost visible line in an edit control.</p> <p>object.emGETFIRSTVISIBLELINE() As Long</p> <p>Return Values The return value is the zero-based index of the uppermost visible line in a multiline edit control. For single-line edit controls, the return value is the zero-based index of the first visible character.</p>
emGETHANDLE	<p>Retrieves a handle of the memory currently allocated for a multiline edit control's text.</p> <p>object.emGETHANDLE() As Long</p> <p>Return Values The return value is a memory handle identifying the buffer that holds the content of the edit control. If an error occurs, such as sending the message to a single-line edit control, the return value is zero.</p>
emGETLIMITTEXT	<p>Retrieves the current text limit, in characters, for an edit control.</p>

object.emGETLIMITTEXT() As Long

Return Values

The return value is the text limit.

emGETLINE

Copies a line of text from an edit control and place it in a specified buffer.

object.emGETLINE(line As Long, lpch As String) As Long

Parameters

line

Specifies the zero-based index of the line to retrieve from a multiline edit control. A value of zero specifies the topmost line. This parameter is ignored by a single-line edit control.

lpch

A string buffer that receives a copy of the line. The first word of the buffer specifies the maximum number of characters that can be copied to the buffer.

Return Values

The return value is the number of characters copied.

The return value is zero if the line number specified by the line parameter is greater than the number of lines in the edit control.

Remarks

The copied line does not contain a terminating null character.

emGETLINECOUNT

Retrieves the number of lines in a multiline edit control.

object.emGETLINECOUNT() As Long

Return Values

The return value is an integer specifying the number of lines in the multiline edit control. If no text is in the edit control, the return value is 1.

emGETMARGINS

Retrieves the widths of the left and right margins for an edit control.

object.emGETMARGINS(wLeft As Integer, wRight As Integer)

Parameters

wLeft

Width of the left margin

wRight

Width of the right margin

emGETMODIFY

Determines whether the content of an edit control has

been modified.

object.emGETMODIFY() As Boolean

Return Values

If the content of edit control has been modified, the return value is TRUE; otherwise, it is FALSE.

Remarks

Windows maintains an internal flag indicating whether the content of the edit control has been changed. This flag is cleared when the edit control is first created; alternatively, an application can send an EM_SETMODIFY message to the edit control to clear the flag.

emGETPASSWORDCHAR Retrieves the password character displayed in an edit control when the user enters text.

object.emGETPASSWORDCHAR() As Integer

Return Values

The return value specifies the character to be displayed in place of the character typed by the user. The return value is 0 if no password character exists.

Remarks

If the edit control is created with the ES_PASSWORD style, the default password character is set to an asterisk (*).

emGETRECT Retrieves the formatting rectangle of an edit control. The formatting rectangle is the limiting rectangle of the text. The limiting rectangle is independent of the size of the edit-control window.

object.emGETRECT(Top As Long, Left As Long, Height As Long, Width As Long)

Parameters

Top, Left, Height, Width

Receives the formatting rectangle.

emGETSEL Gets the starting and ending character positions of the current selection in an edit control.

object.emGETSEL(lpdwStart As Long, lpdwEnd As Long) As String

```
wParam = (WPARAM) (LPDWORD) lpdwStart; //  
receives starting position  
lParam = (LPARAM) (LPDWORD) lpdwEnd; //  
receives ending position
```

Parameters

lpdwStart

Receives the starting position of the selection.

lpdwEnd

Receives the position of the first nonselected character after the end of the selection.

Return Values

The selected text is returned.

emGETTHUMB

Retrieves the position of the scroll box (thumb) in a multiline edit control.

object.emGETTHUMB() As Long

Return Values

The return value is the position of the scroll box.

emHIDeselection

Hides or shows the selection in an edit control.

object.emHIDeselection(fHide As Boolean, fChangeStyle As Boolean)

Parameters

fHide

Value specifying whether to hide or show the selection. If this parameter is zero, the selection is shown. Otherwise, the selection is hidden.

fChangeStyle

Value specifying whether to change the control's esNOHIDeselection window style. If this parameter is zero, the selection is temporarily shown or hidden. Otherwise, the style is changed.

emLIMITTEXT

Limits the amount of text the user may enter into an edit control.

object.emLIMITTEXT(cchMax As Long)

Parameters

cchMax

Specifies the maximum number of characters the user can enter. If this parameter is zero, there is no limit.

Remarks

This method limits only the text the user can enter. It has no effect on any text already in the edit control. If an application places more text into an edit control than is specified in the emLIMITTEXT method, the user can edit the entire contents of the edit control.

The default limit to the amount of text a user can enter in an edit control is 30,000 characters.

emLINEFROMCHAR

Retrieves the index of the line that contains the specified character index in a multiline edit control. A character index is the number of characters from the

beginning of the edit control.

object.emLINEFROMCHAR(ich As Long) As Long

Parameters

ich

Specifies the character index of the character contained in the line whose number is to be retrieved. If the ich parameter is -1, either the line number of the current line (the line containing the caret) is retrieved or, if there is a selection, the line number of the line containing the beginning of the selection is retrieved.

Return Values

The return value is the zero-based line number of the line containing the character index specified by ich.

emLINEINDEX

Retrieves the character index of a line in a multiline edit control. The character index is the number of characters from the beginning of the edit control to the specified line.

object.emLINEINDEX(line As Long) As Long

Parameters

line

Specifies the zero-based line number. A value of -1 specifies the current line number (the line that contains the caret).

Return Values

The return value is the character index of the line specified in the line parameter, or it is -1 if the specified line number is greater than the number of lines in the edit control.

emLINELENGTH

Retrieves the length of a line, in characters, in an edit control.

object.emLINELENGTH(ich As Long) As Long

Parameters

ich

Specifies the character index of a character in the line whose length is to be retrieved. If this parameter is -1, the message returns the number of unselected characters on lines containing selected characters. For example, if the selection extended from the fourth character of one line through the eighth character from the end of the next line, the return value would be 10 (three characters on the first line and seven on the next).

Return Values

The return value is the length, in characters, of the line specified by the ich parameter.

emLINESCROLL

Scrolls the text vertically or horizontally in a multiline edit control.

object.emLINESCROLL(cxScroll As Long, cyScroll As Long) As Boolean

Parameters

cxScroll

Specifies the number of characters to scroll horizontally.

cyScroll

Specifies the number of lines to scroll vertically.

Return Values

If the message is sent to a multiline edit control, the return value is TRUE; if the message is sent to a single-line edit control, the return value is FALSE.

emPOSFROMCHAR

Retrieves the coordinates of the specified character in an edit control.

object.emPOSFROMCHAR(wCharIndex As Integer, xCoord As Integer, yCoord As Integer)

Parameters

wCharIndex

Value of lParam. Specifies the zero-based index of the character.

xCoord, yCoord

Receives the coordinates of the specified character. The coordinates locate the upper-left corner of the character. If the wCharIndex is greater than the index of the last character in the control, the returned coordinates are of the position just past the last character of the control. The coordinates are relative to the upper-left corner of the edit control's client area.

Remarks

For a single-line edit control, the y-coordinate is always zero. A returned coordinate can be negative if the character has been scrolled outside the edit control's client area. The coordinates are truncated to integer values.

emREPLACESEL

Replaces the current selection in an edit control with the specified text.

object.emREPLACESEL(fCanUndo As Boolean, lpszReplace As String)

Parameters

fCanUndo

Specifies whether the replacement operation can be undone. If this is TRUE, the operation can be undone. If this is FALSE, the operation cannot be undone.

lpszReplace
Contains the replacement text.

Remarks
If there is no current selection, the replacement text is inserted at the current location of the caret.

emSCROLL scrolls the text vertically in a multiline edit control.

object.emSCROLL(nScroll As Integer) As Boolean

Parameters
nScroll
Value of wParam. Specifies the action the scroll bar is to take. This parameter may be one of the following values:

Value	Meaning
SB_LINEDOWN	Scrolls down one line.
SB_LINEUP	Scrolls up one line.
SB_PAGEDOWN	Scrolls down one page.
SB_PAGEUP	Scrolls up one page.

Return Values
If the message is successful, the return value is TRUE, otherwise the return value is FALSE.

Remarks
An application should use the EM_LINESCROLL message to scroll to a specific line or character position. An application should use the EM_SCROLLCARET message to scroll the caret into view.

emSCROLLCARET Scrolls the caret into view in an edit control.

object.emSCROLLCARET()

emSETHANDLE Sets the handle of the memory that will be used by a multiline edit control.

object.emSETHANDLE(hloc As Long)

Parameters
hloc
Identifies the memory the edit control uses to store the currently displayed text instead of allocating its own memory. If necessary, the control reallocates this memory.

Remarks
Before an application sets a new memory handle, it should send an EM_GETHANDLE message to retrieve the handle of the current memory buffer and should free that memory.

An edit control automatically reallocates the given buffer whenever it needs additional space for text, or it

removes enough text so that additional space is no longer needed.

Sending an EM_SETHANDLE message clears the undo buffer (EM_CANUNDO returns zero) and the internal modification flag (EM_GETMODIFY returns zero). The edit control window is redrawn.

emSETLIMITTEXT

Sets the text limit for an edit control. The text limit is the maximum amount of text, in bytes, that the edit control can contain.

object.emSETLIMITTEXT(cbMax As Long)

Parameters

cbMax

Specifies the new text limit, in bytes. If this parameter is 0, Windows sets the maximum text limit.

emSETMARGINS

Sets the widths of the left and right margins for an edit control.

object.emSETMARGINS(fwMargin As Long, wLeft As Integer, wRight As Integer)

Parameters

fwMargin

Specifies the margins to set. This parameter can be a combination of the following values:

Value	Meaning
EC_LEFTMARGIN	Sets the left margin.
EC_RIGHTMARGIN	Sets the right margin.
EC_USEFONTINFO	Uses information about the current font of the edit control to set the margins.

wLeft

Specifies the width of the left margin, in pixels.

wRight

Specifies the width of the right margin, in pixels.

emSETMODIFY

Sets or clears the modification flag for an edit control. The modification flag indicates whether the text within the edit control has been modified. It is automatically set whenever the user changes the text.

object.emSETMODIFY(fModified As Integer)

Parameters

fModified

Specifies the new value for the modification flag. A value of TRUE indicates the text has been modified, and a value of FALSE indicates it has not been modified.

emSETPASSWORDCHAR

Sets or removes the password character for a single-line

edit control when the user types text. When a password character is set, that character is displayed in place of each character the user types.

object.emSETPASSWORDCHAR(ch As Integer)

Parameters

ch

Value of wParam. Specifies the character to be displayed in place of the character typed by the user. If this parameter is zero, the characters typed by the user are displayed.

emSETREADONLY

Sets or removes the read-only style of an edit control.

object.emSETREADONLY(fReadOnly As Boolean)
As Boolean

Parameters

fReadOnly

Specifies whether to set or remove the read only style. A value of True sets the read only style; a value of False removes the read only style.

Return Values

If the operation succeeds, the return value is nonzero; otherwise, it is zero.

emSETRECT

Sets the formatting rectangle of a multiline edit control. The formatting rectangle is the limiting rectangle of the text. The limiting rectangle is independent of the size of the edit control window.

This message is processed only by multiline edit controls.

object.emSETRECT(Top As Long, Left As Long,
Height As Long, Width As Long)

Parameters

Top, Left, Height, Width

Specifies the new dimensions of the rectangle.

emSETRECTNP

Sets the formatting rectangle of a multiline edit control.

The emSETRECTNP message is identical to the emSETRECT message, except that the edit control window is not redrawn.

object.emSETRECTNP(Top As Long, Left As Long,
Height As Long, Width As Long)

Parameters

Top, Left, Height, Width

Specifies the new dimensions of the rectangle.

emSETSEL

Selects a range of characters in an edit control.

object.emSETSEL(nStart As Integer, nEnd As Integer)

Parameters

nStart

Value of wParam. Specifies the starting character position of the selection.

nEnd

Specifies the ending character position of the selection.

emSETTABSTOPS

Sets the tab stops in a multiline edit control. When text is copied to the control, any tab character in the text causes space to be generated up to the next tab stop.

This message is processed only by multiline edit controls.

object.emSETTABSTOPS(cTabs As Integer, lpdwTabs As Long) As Boolean

Parameters

cTabs

Specifies the number of tab stops contained in the lpdwTabs parameter. If this parameter is zero, the lpdwTabs parameter is ignored and default tab stops are set at every 32 dialog box units.

lpdwTabs

The first element in an array of Longs specifying the tab stops, in dialog units.

Return Values

If all the tabs are set, the return value is TRUE; otherwise, it is FALSE.

emUNDO

Undo the last edit control operation.

object.emUNDO() As Boolean

Return Values

For a single-line edit control, the return value is always True. For a multiline edit control, the return value is True if the undo operation is successful, or False if the undo operation fails.

Events (Edit Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Event Description	
Click	Standard Event	
DbClick	Standard Event	
KeyDown	Standard Event	
KeyPress	Standard Event	
KeyUp	Standard Event	
MouseDown	Standard Event	
MouseMove	Standard Event	
MouseUp	Standard Event	
ReadyStateChange	Standard Event	

Extended Combo Constituent

Properties

Methods

Events



Class: bbbExCom

Sample-ComboEx\bComboEx.vbg

Overview

The Extended Combo box offers the same features as the Basic Constituents combo box along with some useful additions. The Extended Combo allows for the addition of images within the items displayed in the extended combo. These images are stored in an image list control. The control also allows you to indent the text and images of each item. The following section outlines the custom features of the extended combo box constituent.

Features

In addition to the extended feature offered by the combo box constituent, the extended combo box provides the following methods:

Attach to an Image List control

The cbSETIMAGELIST method sets the image list that the combo box uses; the cbGETIMAGELIST retrieves the handle to the current image list.

Determine if the text in the edit control has changed

To determine if the text has changed, use the cbemHASEDITCHANGED method. This method takes no parameters and returns a boolean indicating if the text has changed.

Prepare to store a large number of items in the combo box

Use the cbINITSTORAGE method before adding a large number of items to the combo box. The lItems parameter sets the number of items to add the combo box and the lMem parameter specifies the length of the strings to be added.

Set the maximum number of character that the user can enter in the edit control

The cbLIMITTEXT method is used to limit the amount of text entered. This method takes one parameter that sets the maximum number of characters that may be entered in the edit portion of the edit combo box.

The following table shows the three combo box types and indicates whether each includes a drop-down list and an edit control:

Combo box type	Drop-down list	Edit control
Drop-down combo box	Yes	Yes
Drop-down list box	Yes	No
Simple combo box	No	Yes

Properties (Extended Combo Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
Appearance	Standard Property	
BackColor	Standard Property	
BorderStyle	Standard Property	
cbsAUTOHSCROLL	Automatically scrolls the text in an edit control to the right when the user types a character at the end of the line. If this style is not set, only text that fits within the rectangular boundary is allowed.	
cbsDISABLENOSCROLL	Shows a disabled vertical scroll bar in the list box when the box does not contain enough items to scroll. Without this style, the scroll bar is hidden when the list box does not contain enough items.	
cbsDROPDOWN	Specifies a drop-down combo box.	
cbsDROPDOWNLIST	Specifies a drop-down list box.	
cbsLOWERCASE	Converts to lowercase all text in both the selection field and the list.	
cbsNOINTEGRAL_HEIGHT	Specifies that the size of the combo box is exactly the size specified by the application when it created the combo box. Normally, Windows sizes a combo box so that it does not display partial items.	
cbsOEMCONVERT	Converts text entered in the combo box edit control from the Windows character set to the OEM character set and then back to the Windows set. This ensures proper character conversion when the application calls the <u>CharToOem</u> function to convert a Windows string in the combo box to OEM characters. This style is most useful for combo boxes that contain filenames and applies only to combo boxes created with the CBS_SIMPLE or CBS_DROPDOWN style.	
cbsOWNERDRAW_FIXED	Specifies that the owner of the list box is responsible for drawing its contents and that the items in the list box are all the same height. The owner window receives a <u>WM_MEASUREITEM</u> message when the combo box is created and a <u>WM_DRAWITEM</u> message when a visual aspect of the combo box has changed.	
cbsOWNERDRAW_VARIABLE	Specifies that the owner of the list box is responsible for drawing its contents and that the items in the list box are variable in height. The owner window receives a <u>WM_MEASUREITEM</u> message for each item in the combo box when you create the combo box and a <u>WM_DRAWITEM</u> message when a visual aspect of the combo box has changed.	
cbsSIMPLE	Specifies a simple combo box.	
cbsSORT	Automatically sorts strings added to the list box.	
cbsUPPERCASE	Converts to uppercase all text in both the selection field and the list.	
CtlHeight dwStyle	Holds the value of the combined bsValues from above. This property is for storage and should not be exposed or	

Enabled	set directly. Standard Property
Font	Standard Property
ForeColor	Standard Property
hWnd	Standard Property
ImageList	Standard Property
ItemData	Standard Property
List	Standard Property
ListCount	Standard Property
ListIndex	Standard Property
MousePointer	Standard Property
Text	Standard Property
TopIndex	Standard Property
WindowProcCB	Sets the address of a callback procedure that will fire for every Windows message. This procedure will be called before default processing. The programmer can change values of parameters or can cancel the message processing altogether.
WindowProcCBAfter	Sets the address of a callback procedure that will fire for every Windows message. This procedure will be called after default processing. The same callback procedure can be used for both WindowProc callbacks.

Methods (Extended Combo Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Method Description	
AboutBox	Invokes the control's About Box	
AddItem	Standard Method	
cbemDELETEITEM	Deletes an item from the list.	
	object.cbemDELETEITEM(Item As Long) As Boolean	
	Parameters	
	Item	
	Specifies the item to delete.	
	Returns	
	Success or failure.	
cbemGETCOMBO_ CONTROL	Gets the hWnd of the Combo control.	
	object.cbemGETCOMBOCONTROL() As Long	
	Returns	
	The handle to the Combo control.	
cbemGETEDIT_ CONTROL	Gets the hWnd of the Edit control within the Combo control.	
	object.cbemGETEDITCONTROL() As Long	
	Returns	
	The handle to the edit control.	
cbemGETEXSTYLE	Gets the extended styles of the combo.	
	object.cbemGETEXSTYLE() As Long	
	Returns	
	The extended style bits.	
cbemGETIMAGELIST	Gets the handle to the image list attached to the combo control.	
	object.cbemGETIMAGELIST() As Long	
	Returns	
	The handle to the associated image list.	
cbemGETITEM	Gets item information from the list.	
	object.cbemGETITEM(mask As Long, item As Long, pszText As String, iImage As Long, iSelectedImage As Long, iOverlay As Long, iIndent As Long, lParam As Long) As Long	
	Parameters	
	mask	
	Determines the parameters that will be filled with information when the method returns. The values can be	

a combination of the following.

CBEIF_TEXT	the psztext parameter will contain valid data.
CBEIF_IMAGE	the iImage parameter will contain valid data.
CBEIF_SELECTEDIMAGE	the iSelectedimage parameter will contain valid data.
CBEIF_OVERLAY	the iOverlay parameter will contain valid data.
CBEIF_INDENT	the IiNDENT parameter will contain valid data.
CBEIF_LPARAM	the lParam parameter will contain valid data.

iItem
Specifies the index of the item.

pszText
The Text of the item.

iImage
Image index within the image list control.

iSelectedImage
Selected Image index within the image list control.

iOverlay
Overlay mask for the item.

iIndent
Amount of indent for the item.

lParam
Program specified data associated with the item.

cbemHASEDIT_CHANGED
Returns
Checks the edit control to see if there has been changes.

object.cbemHASEDIT_CHANGED() As Boolean

cbemINSERTITEM
Returns
True or False.
Inserts an item into the list.

object.cbemINSERTITEM(mask As Long, iItem As Long, pszText As String, iImage As Long, iSelectedImage As Long, iOverlay As Long, iIndent As Long, lParam As Long) As Long

Parameters
mask
Determines the parameters that will be filled with

information when the method returns. The values can be a combination of the following.

CBEIF_TEXT	the psztext parameter will contain valid data.
CBEIF_IMAGE	the iImage parameter will contain valid data.
CBEIF_SELECTEDIMAGE	the iSelectedimage parameter will contain valid data.
CBEIF_OVERLAY	the iOverlay parameter will contain valid data.
CBEIF_INDENT	the iINDENT parameter will contain valid data.
CBEIF_LPARAM	the lParam parameter will contain valid data.

iItem

Specifies the index of the item.

pszText

The Text of the item.

iImage

Image index within the image list control.

iSelectedImage

Selected Image index within the image list control.

iOverlay

Overlay mask for the item.

iIndent

Amount of indent for the item.

lParam

Program specified data associated with the item.

Returns

The index to the item just added.

cbemSETEXSTYLE

Sets the extended styles for the combo.

object.cbemSETEXSTYLE(style As Long) As Long

Parameters

style

The extended style bits.

Returns

The old extended style bits.

cbemSETIMAGELIST

Hooks an image list to the extended combo control.

object.cbemSETIMAGELIST(iImageList As Long) As Long

Parameters
iImage
The handle to the associated image list.

Returns
The old image list handle.
Sets the item information for an existing item in the list.

cbemSETITEM

object.cbemSETITEM(mask As Long, iItem As Long, pszText As String, iImage As Long, iSelectedImage As Long, iOverlay As Long, iIndent As Long, lParam As Long) As Long

Parameters
mask
Determines the parameters that will be filled with information when the method returns. The values can be a combination of the following.

CBEIF_TEXT	the psztext parameter will contain valid data.
CBEIF_IMAGE	the iImage parameter will contain valid data.
CBEIF_SELECTEDIMAGE	the iSelectedimage parameter will contain valid data.
CBEIF_OVERLAY	the iOverlay parameter will contain valid data.
CBEIF_INDENT	the IiNDENT parameter will contain valid data.
CBEIF_LPARAM	the lParam parameter will contain valid data.

iItem
Specifies the index of the item.

pszText
The Text of the item.

iImage
Image index within the image list control.

iSelectedImage
Selected Image index within the image list control.

iOverlay
Overlay mask for the item.

iIndent
Amount of indent for the item.

lParam
Program specified data associated with the item.

cbADDSTRING

Returns
The item index.
Adds a list item.

object.cbADDSTRING(sString As String) As Long

Parameters
sString - specifies the string to add to the combo box.

cbDELETESTRING

Returns
The index of the item added.
Deletes an item from the list.

object.cbDELETEITEM (lItem As Long) As Long

Parameters
lItem - specifies the item to delete.

cbDIR

Returns
index of the deleted item
Adds the filenames matching the specified attributes and path to the list.

object.cbDIR(lAttributes As Long, sDir As String) As Long

Parameters
lAttributes - Specifies the attributes of the files to be added to the list box. This parameter can be a combination of the following values:

Value	Description
DDL_ARCHIVE	Includes archived files.
DDL_DIRECTORY	Includes subdirectories. Subdirectory names are enclosed in square brackets ([]).
DDL_DRIVES	Includes drives. Drives are listed in the form [-x-], where x is the drive letter.
DDL_EXCLUSIVE	Includes only files with the specified attributes. By default, read-write files are listed even if DDL_READWRITE is not specified.
DDL_HIDDEN	Includes hidden files.
DDL_READONLY	Includes read-only files.
DDL_READWRITE	Includes read-write files with no additional attributes.

DDL_SYSTEM Includes system files.

sDir - specifies the filename to add to the list. If the filename contains wildcards (for example, *.*), all files that match the wildcards and have the attributes specified by the lAttributes parameter are added to the list.

Returns
The 0 based index of the last file added to the list.

c7bFINDSTRING Returns the index of the first list item that begins with the specified text.

object.cbFINDSTRING(lIndexStart As Long, sString As String) As Long

Parameters
lIndexStart - specifies the index to begin the search, -1 starts at the beginning.

sString - specifies the string to search for.

Returns
The index of the item matching the string or -1 if none were found.

cbFINDSTRINGEXACT Returns the index of the first list item exactly matching the specified text.

object.cbFINDSTRINGEXACT(lIndexStart As Long, sString As String) As Long

Parameters
lIndexStart - specifies the index to begin the search, -1 starts at the beginning.

sString - specifies the string to search for.

Returns
The index of the item matching the string or -1 if none were found.

cbGETCOUNT Returns the number of list items.

object.cbGETCOUNT() As Long

Returns
the count of items in the list.

cbGETCURSEL Returns the index of the currently selected item, if any.

object.cbGETCURSEL() As Long

Returns
the index of the currently selected item or -1 if none are selected.

cbGETDROPPED_CONTROLRECT Fills the specified rectangle structure with the screen coordinates of a drop-down list.

object.cbGETDROPPEDCONTROLRECT(rect As Long)

cbGETDROPPEDSTATE	<p>Parameters rect - a long representing the first long in a rect structure. This will fill the rest of the rect structure with the values. Returns TRUE if a drop-down list is open; otherwise, it returns FALSE.</p>
	object.cbGETDROPPEDSTATE() As Boolean
cbGETDROPPED_ WIDTH	<p>Returns True if open, otherwise False. Returns the minimum allowable width, in pixels, of the drop down list.</p>
	object.cbGETDROPPEDWIDTH() As Long
cbGETEDITSEL	<p>Returns the minimum width of the dropped list box. Returns the starting and ending position of the current selection. In drop-down list boxes, the window procedure returns an error.</p>
	object.cbGETEDITSEL(lStart As Long, lStop As Long) As Long
	<p>Parameters lStart = carat position within the edit control to begin the selection of text. lStop = carat position within the edit control to end the selection of text.</p>
	<p>Returns zero-based 32-bit value with the starting position of the selection in the low-order word and with the ending position of the first character after the last selected character in the high-order word.</p>
cbGETEXTENDEDUI	<p>Returns TRUE if the combo box is a drop-down combo box or drop-down list box and the extend user-interface flag is set; otherwise, it returns FALSE.</p>
	object.cbGETEXTENDEDUI() As Boolean
	<p>Returns True if using the extended UI, otherwise False. By default, the F4 key opens or closes the list and the DOWN ARROW changes the current selection. In a combo box with the extended user interface, the F4 key is disabled and pressing the DOWN ARROW key opens the drop-down list.</p>
cbGETHORIZONTAL_ EXTENT	<p>Returns the scrollable width, in pixels, of the drop down list.</p>
	object.cbGETHORIZONTALALEXTENT() As Long

cbGETITEMDATA Returns the scrollable width, in pixels.
Returns the 32-bit value associated with the specified list item.

object.cbGETITEMDATA(IIndex As Long) As Long

Parameters
IIndex - specifies the item.

cbGETITEMHEIGHT Returns the 32 bit “item data” value. This value is set though cbSETITEMDATA.
Returns the height, in pixels, of the specified owner-drawn list item.

object.cbGETITEMHEIGHT(IIndex As Long) As Long

Parameters
IIndex - this parameter must be -1 to retrieve the height of the selection field. It must be zero to retrieve the height of list items, unless the combo box has the cbsOWNERDRAWVARIABLE style. In that case, the index parameter is the zero-based index of a specific list item.

cbGETLBTEXT Returns the height of the specified item.
Copies the specified list text to the specified buffer.

object.cbGETLBTEXT(IIndex As Long) As String

Parameters
IIndex - specifies the item from which to retrieve the text.

cbGETLBTEXTLEN Returns the text of the item.
Returns the length, in bytes, of the specified list text.

object.cbGETLBTEXTLEN(IIndex As Long) As Long

Parameters
IIndex - specifies the item from which to retrieve the text length.

cbGETLOCALE Returns the length of the text of the item.
Returns the current locale for the list.

object.cbGETLOCALE() As Long

cbGETTOPINDEX Returns the index of the first visible item in the drop down list.

object.cbGETTOPINDEX() As Long

cbINITSTORAGE Initializes space for the specified number of items and the

specified number of bytes for item strings.

object.cbINITSTORAGE(IItems As Long, IMem As Long)
As Long

Parameters

IItems - specifies the number of items to add

IMem - specifies the amount of memory to allocate for strings.

Returns

maximum number of items that the memory object can store.

cbINSERTSTRING

Inserts a list item at the specified position.

object.cbINSERTSTRING(IIndex As Long, sString As String) As Long

Parameters

IIndex - the 0 based index that indicates the point at which to insert the string. If -1 is specified, the string is inserted at the end of the list.

sString - specifies the string to insert.

Returns

the index at the point the string was inserted.

cbLIMITTEXT

Sets the maximum number of characters a user can enter in the edit control. In drop-down list boxes, the window procedure returns an error.

object.cbLIMITTEXT(ILimit As Long)

Parameters

ILimit - specifies the maximum number of characters that can be typed into the edit portion of a combo box.

cbRESETCONTENT

Removes the contents of the list.

object.cbRESETCONTENT()

cbSELECTSTRING

Selects the first list item, if any, that begins with the characters in the specified text.

object.cbSELECTSTRING(IIndex As Long, sString As String) As Long

Parameters

IIndex - indicates the position within the list to begin the search. Setting this -1 will begin the search at the beginning of the list.

sString - specifies the prefix string to search for.

Returns

The index of the item selected or -1 if the item was not found.

cbSETCURSEL	<p>Sets the current selection.</p> <p>object.cbSETCURSEL(IIndex As Long) As Long</p> <p>Parameters IIndex - index of the item to select or -1 to deselect all items.</p> <p>Returns the item selected or -1 if none selected.</p>
cbSETDROPPEDWIDTH	<p>Sets the minimum allowable width, in pixels, of the drop down list.</p> <p>object.cbSETDROPPEDWIDTH(IWidth As Long) As Long</p> <p>Parameters IWidth - specifies the minimum width allowable for the drop list.</p> <p>Returns the new width of the list box or -1 if it fails.</p>
cbSETEDITSEL	<p>Selects the specified range of text. In drop-down list boxes, the window procedure returns an error.</p> <p>object.cbSETEDITSEL(IStart As Long, IStop As Long) As Long</p> <p>Parameters Istart - specifies the starting position of selected text. Istop - specifies the ending position of selected text.</p> <p>Returns If successful 1 is returned, if it fails -1 is returned.</p>
cbSETEXTENDEDUI	<p>Sets or clears the extended user-interface flag. This flag changes the keys that open and close the list in a drop-down combo box or drop-down list box. If the combo box is a simple combo box, the window procedure returns an error.</p> <p>object.cbSETEXTENDEDUI(bUI As Boolean) As Boolean</p> <p>Parameters bUI - if True, turns on the extended UI, if False turn it off.</p> <p>Returns Success or failure.</p>
cbSETHORIZONTAL_EXTENT	<p>Sets the scrollable width, in pixels, of the drop down list.</p> <p>object.cbSETHORIZONTAL_EXTENT(IWidth As Long)</p> <p>Parameters IWidth - specifies the drop down list width.</p>
cbSETITEMDATA	<p>Associates the specified 32-bit value with a list item.</p>

object.cbSETITEMDATA(IIndex As Long, IData As Long)
As Boolean

Parameters
IIndex - specifies the item to associate the IData value with.

IData - the value to associate with the specified ite.

Returns
Success or failure.

cbSETITEMHEIGHT
Sets the height of the specified owner-drawn list item or the selection field.

object.cbSETITEMHEIGHT(IIndex As Long, IHeight As Long) As Boolean

Parameters
IIndex - specifies the item.

IHeight - specifies the height

Returns
Success or failure

cbSETLOCALE
Sets the current locale for the list. The locale affects how the list is sorted if it has the cbsSORT style and strings are added using cbADDSTRING.

object.cbSETLOCALE(ILocale As Long) As Long

Parameters
ILocale - specifies the new locale

Returns
the old locale.

cbSETTOPINDEX
Scrolls the drop down list so the specified item is at the top of the visible range.

object.cbSETTOPINDEX(IIndex As Long) As Boolean

Parameters
IIndex - species the 0 based item.

Returns
Success or Failure

cbSHOWDROPDOWN
Shows or hides the drop-down list. This message has no effect on simple combo boxes.

object.cbSHOWDROPDOWN(bShow As Boolean)

Parameters
bShow - if True, shows the drop down list, if False hides the drop down list.

Clear
Standard Method

FontSelect
Refresh
ReleaseFont
RemoveItem

Standard Method

Standard Method

Events (Extended Combo Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Event Description	
Click	Standard Event	
Command	Processes notification messages from the edit control and list window and sends corresponding combo box notification messages to the parent window. For edit control notifications, the window procedure may update the list window's current selection, caret index, and top index. For list notification messages, the window procedure may update the content of the selection field.	
	CBN_CLOSEUP	Indicates the list in a drop-down combo box or drop-down list box is about to close.
	CBN_DBLCLK	Indicates the user has double-clicked a list item in a simple combo box.
	CBN_DROPDOWN	Indicates the list in a drop-down combo box or drop-down list box is about to open.
	CBN_EDITCHANGE	Indicates the user has changed the text in the edit control of a simple or drop-down combo box. This notification message is sent after the altered text is displayed.
	CBN_EDITUPDATE	Indicates the user has changed the text in the edit control of a simple or drop-down combo box. This notification message is sent before the altered text is displayed.
	CBN_ERRSPACE	Indicates the combo box cannot allocate enough memory to carry out a request, such as adding a list item.
	CBN_KILLFOCUS	Indicates the combo box is about to lose the input focus.
	CBN_SELCHANGE	Indicates the current selection has changed.
	CBN_SELENDCANCEL	Indicates that the selection made in the drop down list, while it was dropped down, should be ignored.
	CBN_SELENDOK	Indicates that the selection made drop down list, while it was dropped down, should be

	CBN_SETFOCUS	accepted. Indicates the combo box has received the input focus.
DbIClick	Standard Event	
KeyDown	Standard Event	
KeyPress	Standard Event	
KeyUp	Standard Event	
MouseDown	Standard Event	
MouseMove	Standard Event	
MouseUp	Standard Event	
Resize		

CALL BACKS

Event Name

Event Description

WindowProc

A callback procedure used to respond to or process any Windows message. The extended combo control is already subclassed for you eliminating the need to write or use a subclassing control to get to any Windows' message.

Sub WinProc(mycom As Object, msg As Long, wp As Long, lp As Long, _ bDone As Long)

mycom the control this message belongs to.
You can access the value of the properties from this Object

msg the message value, WM_...

wp the WPARAM value for the message.

lp the LPARAM value for the message.

bDone a flag that can be set to skip processing this message, also indicates before or after processing on entry.

Notice that the WinProc contains essentially the same information as any Windows message. The typical hWnd parameter has been replaced with the control and the bDone parameter has been added. It is possible and may be necessary to do some creative programming to get the value that is represented by the lp parameter as this value could be a direct long value or a pointer to almost any type of data. bDone is set to FALSE on entry to the WinProc Sub when firing the event for before default processing.

It will be TRUE when the routine is processing the message after default processing. When bDone is FALSE, it may be set to TRUE before exiting the routine to skip default processing. When this is done, the “after” event will not be fired.

Header Constituent

Properties

Methods

Events



Class: bbbHead

Sample-Header\bHead.vbg

Overview

A header is a control that resembles one or more buttons that are connected in a horizontal row. The buttons are commonly placed above list boxes to allow columns of the list box to be sorted. The Basic Constituents Header allows you to create as many header items as you like. A header items can contain text, which can be aligned to the left, center or right, and may also contain a bitmap (stdPicture object). You can add and delete items at run-time, retrieve their bounding rectangles and get the count of items in the header. The following section outlines some of these features in detail.

Features

This section outlines some of the features of the Basic Constituents Header Control.

Add a new item to the header

To add a new item, use the hdmINSERTITEM method. This method takes parameters that set the index of the item, its type, the text of the item, its picture, etc. See the reference for more information on the parameters.

The following lines of code add a new item to the header. This item displays the text “Header” and the text is centered. The 2nd parameter indicates that the header will display text. Even though we are only displaying text, we still declare an instance of a StdPicture as a place holder for the bitmap:

```
Dim bRet As Boolean
Dim iPicture As StdPicture
bRet = Bbbhead1.hdmINSERTITEM(0, HDI_TEXT, 100, ByVal "Header", iPicture,
    HDF_CENTER Or HDF_STRING, 0)
bRet = Bbbhead1.hdmINSERTITEM(1, HDI_TEXT, 100, ByVal "Header", iPicture,
    HDF_CENTER Or HDF_STRING, 0)
```

The following screen shows the result of the operation. This form contains an ActiveX control that uses a header as a constituent:



Figure 12: The Basic Constituents Header control displaying two items (button style with text).

Get information about an item in the header control

Use the hdmGETITEM method to retrieve information about an item in the header control. See the reference section later in this chapter for more information about the parameters to this method.

Determine the number of items in a header

The hdmGETITEMCOUNT method returns the number of items in a header. This method requires no parameters and returns the number of items in the header.

Retrieve the size of and dimensions of an item in the header

The `hdmLAYOUT` method is used to determine the bounding rectangle, as well as the position of an item in the header. This is particularly useful if you are inserting an item between 2 headers. For example, imagine that you have 2 items in a header (A and B) and are preparing to insert a third, C. In reality, you want to insert C between A and B. To do this, you need to know the position and size of item B. When C is inserted (in B's place), it needs to be set to the same size and position as B. The `hdmLAYOUT` method provides the information that you need to do this. See the reference section later in this chapter for more information about this method.

Track the actions taken on the divider between 2 columns

The `BeginTrack` event is triggered when a user drags the divider between 2 headers. The `EndTrack` event is fired when the dragging stops. The `DividerClick` and `DividerDbClick` events are triggered when the user clicks or double-clicks a divider.

Properties (Header Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
BackColor	Standard Property	
BorderStyle	Standard Property	
ecsADJUSTABLE	Enables / disables dragging on the toolbar	
Container	Standard Property	
DragIcon	Standard Property	
DragMode	Standard Property	
Enabled	Standard Property	
ForeColor	Standard Property	
Height	Standard Property	
HelpContextID	Standard Property	
hdsBUTTONS	Header items behave like buttons.	
hdsHIDDEN	Indicates a header control that is intended to be hidden. This style does not hide the control; instead, it causes the header control to return zero in the cy member of the <u>WINDOWPOS</u> structure returned by an <u>HDM_LAYOUT</u> message. You would then hide the control by setting its height to zero.	
hdsHORZ	Header style, if true, The header control is horizontal.	
hWnd	Standard Property	
Index	Standard Property	
Left	Standard Property	
MouseIcon	Standard Property	
MousePointer	Standard Property	
Parent	Standard Property	
TabIndex	Standard Property	
TabStop	Standard Property	
Tag	Standard Property	

Top	Standard Property
Visible	Standard Property
WhatsThisHelpID	Standard Property
Width	Standard Property

Methods (Header Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>												
Method Name	Method Description													
hdmDELETEITEM	<p>Deletes an item from a header control.</p> <p>object.hdmDELETEITEM(index As Long) As Boolean</p> <p>Parameters index Index of the item to delete.</p> <p>Return Values If the operation succeeds, the return value is TRUE. If the operation fails, the return value is FALSE.</p>													
hdmGETITEM	<p>Retrieves information about an item in a header control.</p> <p>object.hdmGETITEM (index As Long, mask As Long, cxy As Long, pszText As String, fmt As Long, lParam As Long) As Boolean</p> <p>Parameters index Index of the item for which information is to be retrieved. Mask Mask flags that indicate which of the other structure members contain valid data. This member can be a combination of the following values:</p> <table border="0"> <tr> <td>HDI_BITMAP</td> <td>The hbm member is valid.</td> </tr> <tr> <td>HDI_FORMAT</td> <td>The fmt member is valid.</td> </tr> <tr> <td>HDI_HEIGHT</td> <td>The cxy member is valid and specifies the height of the item.</td> </tr> <tr> <td>HDI_LPARAM</td> <td>The lParam member is valid.</td> </tr> <tr> <td>HDI_TEXT</td> <td>The pszText and cchTextMax members are valid.</td> </tr> <tr> <td>HDI_WIDTH</td> <td>The cxy member is valid and specifies the width of the item.</td> </tr> </table> <p>cxy - Width or height of item.</p> <p>PszText - Pointer to item string.</p>		HDI_BITMAP	The hbm member is valid.	HDI_FORMAT	The fmt member is valid.	HDI_HEIGHT	The cxy member is valid and specifies the height of the item.	HDI_LPARAM	The lParam member is valid.	HDI_TEXT	The pszText and cchTextMax members are valid.	HDI_WIDTH	The cxy member is valid and specifies the width of the item.
HDI_BITMAP	The hbm member is valid.													
HDI_FORMAT	The fmt member is valid.													
HDI_HEIGHT	The cxy member is valid and specifies the height of the item.													
HDI_LPARAM	The lParam member is valid.													
HDI_TEXT	The pszText and cchTextMax members are valid.													
HDI_WIDTH	The cxy member is valid and specifies the width of the item.													

fmt - A set of bit flags that specify the item's format. This member can include one of the following text justification or right-to-left reading order bit flags:

HDF_CENTER	Centers the contents of the item.
HDF_LEFT	Left aligns the contents of the item.
HDF_RIGHT	Right aligns the contents of the item.
HDF_RTLREADING	Displays text using right-to-left reading order.

The preceding value is combined with one of the following values:

HDF_BITMAP	The item displays a bitmap.
HDF_OWNERDRAW	The owner window of the header control draws the item.
HDF_STRING	The item displays a string.

The HDF_JUSTIFYMASK mask will isolate the text justification portion fmt.

IParam - Application-defined item data.

Return Values

If the operation succeeds, the return value is TRUE.
If the operation fails, the return value is FALSE.

hdmGETITEMCOUNT Retrieves a count of the items in a header control.

hdmGETITEMCOUNT () As Long

Return Values

If the operation succeeds, the return value is the number of items.

If the operation fails, the return value is - 1.

hdmHITTEST Tests a point to determine which header item, if any, is at the specified point.

object.hdmHITTEST (x As Long, y As Long, flags As Long) As Long

Parameters

x, y
Points to test, in client coordinates

flags

Variable that receives information about the results of a hit test. This member can be one or more of the following values:

HHT_NOWHERE	The point is inside the bounding rectangle of the header control but is not over a header item.
HHT_ONDIVIDER	The point is on the divider between two header items.
HHT_ONDIVOPEN	The point is on the divider of an item that has a width of zero. Dragging the divider reveals the item instead of resizing the item to the left of the divider.
HHT_ONHEADER	The point is inside the bounding rectangle of the header control.
HHT_TOLEFT	The point is to the left of the bounding rectangle of the header control.
HHT_TORIGHT	The point is to the right of the bounding rectangle of the header control.

Two of these values can be combined, such as when the position is above and to the left of the client area.

Return Values

Returns the index of the item at the specified position, if any, or - 1 otherwise.

hdmINSERTITEM

Inserts a new item into a header control.

object.hdmINSERTITEM (index As Long, mask As Long, cxy As Long, pszText As String, hbm As StdPicture, fmt As Long, lParam As Long) As Boolean

Parameters

index

Index of the item after which the new item is to be inserted. The new item is inserted at the end of the header control if index is greater than or equal to the number of items in the control. If index is zero, the new item is inserted at the beginning of the header control.

mask

Mask flags that indicate which of the other structure members contain valid data. This member can be a

combination of the following values:

HDI_BITMAP	The hbm member is valid.
HDI_FORMAT	The fmt member is valid.
HDI_HEIGHT	The cxy member is valid and specifies the height of the item.
HDI_LPARAM	The lParam member is valid.
HDI_TEXT	The pszText and cchTextMax members are valid.
HDI_WIDTH	The cxy member is valid and specifies the width of the item.

cxy
Width or height of item.

pszText
Pointer to item string.

hbm
Handle to item bitmap.

fmt
A set of bit flags that specify the item's format. This member can include one of the following text justification or right-to-left reading order bit flags:

HDF_CENTER	Centers the contents of the item.
HDF_LEFT	Left aligns the contents of the item.
HDF_RIGHT	Right aligns the contents of the item.
HDF_RTLREADING	Displays text using right-to-left reading order.

The preceding value is combined with one of the following values:

HDF_BITMAP	The item displays a bitmap.
HDF_OWNERDRAW	The owner window of the header control draws the item.
HDF_STRING	The item displays a

string.

The HDF_JUSTIFYMASK mask will isolate the text justification portion fmt.

IParam

Application-defined item data.

Return Values

If the operation succeeds, the return value is the index of the new item.

If the operation fails, the return value is - 1.

hdmLAYOUT

Message retrieves the size and position of a header control within a given rectangle. This message is used to determine the appropriate dimensions for a new header control that is to occupy the given rectangle.

object.hdmLAYOUT (iLeft As Long, iTop As Long, iRight As Long, iBottom As Long, hWnd As Long, hwndInsertAfter As Long, x As Long, y As Long, cx As Long, cy As Long, flags As Long) As Boolean

Parameters

hWnd

Identifies the window.

hwndInsertAfter

Specifies the position of the window in Z order (front-to-back position). This member can be the handle of the window behind which this window is placed, or can be one of the special values listed with the [SetWindowPos](#) function.

x, y

Specifies the window position.

cx, cy

Specifies the window size.

flags

Specifies the window position. This member can be one of the following values:

Return Values

If the operation succeeds, the return value is TRUE.

If the operation fails, the return value is FALSE.

hdmSETITEM

Sets the attributes of the specified item in a header control.

object.hdmSETITEM (index As Long, mask As Long, cxy As Long, pszText As String, hbm As StdPicture, fmt As Long, IParam As Long) As Boolean

Parameters

index - Index of the item whose attributes are to be changed.

mask - Mask flags that indicate which of the other structure members contain valid data. This member can be a combination of the following values:

HDI_BITMAP	The hbm member is valid.
HDI_FORMAT	The fmt member is valid.
HDI_HEIGHT	The cxy member is valid and specifies the height of the item.
HDI_LPARAM	The lParam member is valid.
HDI_TEXT	The pszText and cchTextMax members are valid.
HDI_WIDTH	The cxy member is valid and specifies the width of the item.

cxy - Width or height of item.

pszText - Pointer to item string.

hbm - Handle to item bitmap.

fmt - A set of bit flags that specify the item's format.

This member can include one of the following text justification or right-to-left reading order bit flags:

HDF_CENTER	Centers the contents of the item.
HDF_LEFT	Left aligns the contents of the item.
HDF_RIGHT	Right aligns the contents of the item.
HDF_RTLREADING	Displays text using right-to-left reading order.

The preceding value is combined with one of the following values:

HDF_BITMAP	The item displays a bitmap.
HDF_OWNERDRAW	The owner window of the header control draws the item.
HDF_STRING	The item displays a string.

The HDF_JUSTIFYMASK mask will isolate the text justification portion fmt.

lParam

Application-defined item data.

Return Values

If the operation succeeds, the return value is TRUE.

If the operation fails, the return value is FALSE.

Remarks

The HDN_ITEMCHANGING notification message is sent to the parent window before the item attributes are changed. The parent window can return FALSE to prevent the changes, and in that case, HDM_SETITEM returns FALSE. If the parent window returns TRUE, the changes are made and the parent window receives the HDN_ITEMCHANGED notification message.

Events (Header Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Event Description	
BeginTrack	Fires when a user begins dragging a column divider. Sub BeginTrack(Item As Long, Cancel As Boolean) Parameters: Item Column being dragged. Cancel - if set to True rejects the column divider drag.	
DividerDbClick	Fires when a divider is double clicked. Sub DividerDbClick(Item As Long) Parameters: Item - column double clicked.	
EndTrack	Fires when the user stops dragging an item. Sub EndTrack(Item As Long) Parameters: Item - column that was being dragged.	
ItemClick	Fires when the user clicks on a header item. Sub ItemClick(Item As Long) Parameters: Item - header item clicked.	
ItemDbClick	Fires when the user double clicks on a header item. Sub ItemDbClick(Item As Long) Parameters: Item - header item double clicked.	

CALL BACKS

Call Back Name	Event Description
-----------------------	--------------------------

Notify

The notify callback is a prototype for a call back procedure. It is called if the NotifyProcCB property is set to a function address. Some of the messages available though the notify call back also have corresponding events. It is possible to use either or both, however we do not fully document the Notify messages here. See the WIN32 documentation for WM_NOTIFY as it applies to the header control. The following notification messages are available:

HDN_BEGINTRACK
HDN_DIVIDERDBLCLICK
HDN_ENDTRACK
HDN_ITEMCHANGED
HDN_ITEMCHANGING
HDN_ITEMCLICK
HDN_ITEMDBLCLICK
HDN_TRACK

Syntax:

Sub Notify(myControl As Object, wp As Long, lp As Long)

Parameters:

myControl :A reference to the header control.

wp :The WPARAM parameter associated with the notification message.

lp :The LPARAM parameter associated with the notification message.

DrawItem

DrawItem is a prototype for a call back procedure. It does not have to be named DrawItem, but it must have two parameters with the correct type and be a SUB (not a function returning a value).

You place the address of this routine into the ItemDrawCB property and set the cbOWNERDRAW property to True and this routine will be called to perform the painting of the button.

Sub DrawItem (objItem As Object, dw As DRAWITEMSTRUCT)

objItem the control being painted. You can access the value of the properties from this Object.

dw is a draw item structure as used in all owner draw controls in Windows.

WindowProc

A callback procedure used to respond to or process any Windows message. The header control is already subclassed for you eliminating the need to write or use a subclassing control to get to any Windows' message.

Sub WinProc(myhed As Object, msg As Long, wp As Long, lp As Long, _ bDone As Long)

myhed the control this message belongs to. You can access the value of the properties from this Object

msg the message value, WM_...

wp the WPARAM value for the message.

lp the LPARAM value for the message.

bDone a flag that can be set to skip processing this message,
 also indicates before or after processing on entry.

Notice that the WinProc contains essentially the same information as any Windows message. The typical hWnd parameter has been replaced with the control and the bDone parameter has been added. It is possible and may be necessary to do some creative programming to get the value that is represented by the lp parameter as this value could be a direct long value or a pointer to almost any type of data. bDone is set to FALSE on entry to the WinProc Sub when firing the event for before default processing. It will be TRUE when the routine is processing the message after default processing. When bDone is FALSE, it may be set to TRUE before exiting the routine to skip default processing. When this is done, the "after" event will not be fired.

Image List Constituent

Properties

Methods

Events



Class: bbbImgli

Sample-ImgList\blist.vbg

Overview

An *image list* is a control that stores a collection of same-sized images, each of which can be referred to by its index. Image lists are used to manage a set of icons or bitmaps.

The BeCubed image list allows you to add and remove items as well as draw an item to a device context (hDC).

Properties (Image List Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property	Description	
BackColor	Standard Property	
hImageList	Returns a handle to the image list.	
ImageHeight	Sets or returns the height of an image. This cannot be set after adding images.	
ImageWidth	Sets or returns the width of an image. This cannot be set after adding images.	
Index	Standard Property	
Name	Standard Property	
Parent	Standard Property	
Tag	Standard Property	

Methods (Image List Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method	Description	
AboutBox	Invokes the control's About Box.	
Add	Adds pictures to an image view. Pictures must be bitmaps, not icons or metafiles. object.Add(pic As StdPicture, picmask As StdPicture)	
	Parameters	
	pic Specifies the picture to add.	
	picmask Specifies the picture mask to add.	
	Return Value (Boolean)	
	Returns the index of the picture if successful.	
Draw	Draws a picture from an image list to the specified device context. object.Draw(index As Long, hDC As Long, x As Long, y As Long, Style As Long)	
	Parameters	
	index Specifies the index of the picture to draw.	
	hDC Specifies the device context to draw the picture.	
	x, y Specifies the top, left coordinate of the picture within the hDC, expressed in pixels, based on window coordinates.	
	style Specifies how the picture is rendered using one of the following options:	
	ILD_FOCUS	Draws the image in a manner that appears to have the focus.
	ILD_SELECTED	Draws the image in a manner that appears to be selected.
	ILD_BLEND	Draws the image so that it appears disabled.

ILD_BLEND25 Draws the image, blending 25 percent with the system highlight color. This value has no effect if the image list does not contain a mask.

ILD_BLEND50 Draws the image, blending 50 percent with the system highlight color. This value has no effect if the image list does not contain a mask.

ILD_MASK Draws the mask.

ILD_NORMAL Draws the image using the background color for the image list. If the background color is the CLR_NONE value, the image is drawn transparently using the mask.

ILD_TRANSPARENT Draws the image transparently using the mask, regardless of the background color. This value has no effect if the image list does not contain a mask.

Return Value (Boolean)

Returns the index of the picture if successful.

Removes pictures from an image view.

Remove

object.Remove(index As Long)

Parameters

index

Specifies the index of the picture to draw.

Return Value (Boolean)

There is no return value.

Events (Image List Constituent)

Description

Properties

Methods

There are no events for this control.

List Constituent

Properties

Methods

Events



Class: `bbList` `Sample-List\bbList.vbg`

Overview

The list constituent is the BeCubed version of the standard list box. This list box contains a great amount of functionality not contained within its standard control counterpart. The list constituent allows you add a list of files to the list box, allows you to search for strings and supports owner-draw capabilities. The following section outlines some of the custom features of the list box constituent.

Features

For ease of readability, this section only outlines features that are not contained within the standard list box control.

Add a list of file names to the list box

The `lbDIR` method adds the list of filenames specified in the `sDir` parameter. The `lAttributes` parameter is used to set the type of files that the list should contain (list of drives, list of directories, read only, hidden, system, etc.).

Search for a string in the list box

The `lbFINDSTRING` and `lbFINDSTRINGEXACT` methods allow you to search for strings in the list box. The `lbFINDSTRING` method searches for the first item that begins with the string searched for. The method takes 2 parameters: where in the list to start (-1 begins at the top of the list) and the string to search for. The `lbFINDSTRINGEXACT` searches for a string that exactly matches the string searched for. The following line of code searches a list constituent for an exact match on the string "Hello":

```
lRet = BBList1.lbFINDSTRINGEXACT(-1, "Hello")
```

If found, `lRet` contains the index of the item.

Determine the height of an item in the list box

The `lbGETITEMHEIGHT` method returns the height of an item. This method takes one parameter that contains the index of the item to check. The return value is the height, in pixels, of the item. The following line of code retrieves the height of the 2nd item in the list box:

```
lHeight = BBList1.lbGETITEMHEIGHT(2)
```

Retrieve the bounding rectangle for a specific item in the list

The `lbGETITEMRECT` method is used to retrieve the bounding rectangle of an item. This method takes 2 parameters: The index of the item to check and a rectangle structure that receives the coordinates.

Retrieve the index of the first visible item in the list box

The `lbGETTOPINDEX` method is used to determine the first item that is visible in the list box. This method takes no parameters and returns the index of the first visible item:

```
lIndex = BBListBox1.lbGETTOPINDEX
```

Prepare to store a large number of items to the list box

The `lbINITSTORAGE` method is used to prepare to store a large number of items. This method takes 2 parameters: The number of items to be stored and the length of the items. The following line of code attempts to add memory for 1000 items, each 10 positions wide. If the return is -2, not enough memory is available:

```
lRet = BBListBox1.lbINITSTORAGE(1000, 10)
```

Retrieve the index closest to a specific point on the list box

The `lbITEMFROMPOINT` method can be used to determine the item that is closest to a specific point on the list box. The method takes 2 parameters: The x and y coordinates of the point. Both of these positions are relative to the

upper-left corner of the client area of the list box.

Programmatically select a range of items in a multi-selection list box

The `lbSELITEMRANGE` method selects a range of items. This method takes 3 parameters. The first determines whether or not to highlight the strings, the second and third set the starting and ending positions for the selection. The following line of code selects and highlights the first 10 items of a list box:

```
lRet = BBListBox1.lbSETITEMRANGE(True, 0, 9)
```

Set tab stops in the list box

The `lbSETTABSTOPS` method is used to set tab stops within the list box. This method takes 2 parameters. The first is the number of tab stops, the second is an array that sets the position of the tab stops.

See the `lbSETTABSTOPS` method in the reference section for more information about this method.

Scroll the list box to place a specific item at the top

The `lbSETTOPINDEX` method can be used to set a specific item at the top of the range of visible items.

Properties (List Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
Appearance	Standard Property	
BackColor	Standard Property	
dwStyle	Holds the value of the combined bsValues from above. This property is for storage and should not be exposed or set directly.	
Enabled	Standard Property	
Font	Standard Property	
ForeColor	Standard Property	
hWnd	Standard Property	
ItemCompareCB	Enables sorting compare routine.	
ItemData	sets or returns the number associated with a particular line item	
ItemDrawCB	sets the address of the callback procedure when programming an owner draw button. This must be a procedure in a BSA module with a particular set of parameters. See the ButtonDraw documentation for details.	
ItemMeasureCB	Enables mixed line height list items.	
lbsDISABLENO_SCROLL	Shows a disabled vertical scroll bar for the list box when the box does not contain enough items to scroll. If you do not specify this style, the scroll bar is hidden when the list box does not contain enough items.	
lbsEXTENDEDSEL	Allows multiple items to be selected by using the SHIFT key and the mouse or special key combinations.	
lbsHASSTRINGS	Specifies that a list box contains items consisting of strings. The list box maintains the memory and addresses for the strings so that the application can use the <code>lbGETTEXT</code> method to retrieve the text for a particular item. By default, all list boxes except owner-drawn list boxes have this style. You can create an owner-drawn list box either with or without this style.	
lbsMULTICOLUMN	Specifies a multicolumn list box that is scrolled horizontally. The <code>lbSETCOLUMNWIDTH</code> message sets the width of the columns.	
lbsMULTIPLESEL	Turns string selection on or off each time the user clicks or double-clicks a string in the list box. The user can select any number of strings.	

lbsNOINTEGRAL_HEIGHT	Specifies that the size of the list box is exactly the size specified by the application when it created the list box. Normally, Windows sizes a list box so that the list box does not display partial items.
lbsNOSEL	Specifies that the list box contains items that can be viewed but not selected.
lbsNOTIFY	Notifies the parent window with an input message whenever the user clicks or double-clicks a string in the list box.
lbsOWNERDRAW_FIXED	Specifies that the owner of the list box is responsible for drawing its contents and that the items in the list box are the same height. The owner window receives an ItemMeasure call back when the list box is created and a ItemDraw call back when a visual aspect of the list box has changed.
lbsOWNERDRAW_VARIABLE	Specifies that the owner of the list box is responsible for drawing its contents and that the items in the list box are variable in height. The owner window receives a ItemMeasure call back for each item in the combo box when the list box is created and a DrawItem call back when a visual aspect of the list box has changed.
lbsSORT	Sorts strings in the list box alphabetically.
lbsUSETABSTOPS	Enables a list box to recognize and expand tab characters when drawing its strings. The default tab positions are 32 dialog box units. A dialog box unit is a horizontal or vertical distance. One horizontal dialog box unit is equal to 0.25 of the current dialog box base-width unit. Windows calculates these units based on the height and width of the current system font. The GetDialogBaseUnits function returns the current dialog box base units in pixels.
lbsWANTKEYBOARD_INPUT	Specifies that the owner of the list box receives WM_VKEYTOITEM messages whenever the user presses a key and the list box has the input focus. This enables an application to perform special processing on the keyboard input.
List	Returns or sets the items contained in a control's list portion. The list property is a string array.
ListCount	number of items in the control
ListIndex	item number of the currently selected items (-1 nothing selected)
MousePointer	Standard Property
MouseIcon	Standard Property
SelCount	number of items currently selected.
Selected	boolean, sets or returns the selected state of an individual items.
Text	string contents of the currently selected item
TopIndex	index number of the item at the top of the lists' view window.
WindowProcCB	Sets the address of a callback procedure that will fire for every Windows message. This procedure will be called before default processing. The programmer can change values of parameters or can cancel the message processing altogether.
WindowProcCBAfter	Sets the address of a callback procedure that will fire for

every Windows message. This procedure will be called after default processing. The same callback procedure can be used for both WindowProc callbacks.

Methods (List Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Method Description	
AboutBox	Standard Method	
AddItem	Standard Method	
Clear	Standard Method	
ColorTranslate	Translates a system color into an RGB (can also pass just the RGB)	
FontSelect	Enables the use of a font other than the standard default font. Useful for ownerdraw listbox.	
lbADDFILE	<p>Inserts a file into a directory list box filled by the <u>DlgDirList</u> function and retrieves the list box index of the inserted item.</p> <p>object.lbADDFILE(sFile As String) As Long</p> <p>Parameters sFile The name of the file to add.</p> <p>Return Values The return value is the zero-based index of the file that was added, or -1 if an error occurs.</p>	
lbADDSTRING	<p>Adds a string to a list box and returns its index.</p> <p>object.lbADDSTRING(sString As String) As Long</p> <p>Parameters sString String that is to be added.</p> <p>If you create the list box with an owner-drawn style but without the lbsHASSTRINGS style, You should use the lbGETITEMDATA and lbSETITEMDATA messages to retrieve or modify the item data after adding an item to the list.</p> <p>Return Values The return value is the zero-based index of the string in the list box. If an error occurs, the return value is -1. If there is insufficient space to store the new string, the return value is -2.</p> <p>Remarks If you create an owner-drawn list box with the lbsSORT style but not the lbsHASSTRINGS style, you must use the ItemCompare call back to sort the items in the list.</p>	
lbDELETESTRING	Removes a string from a list box and returns the number of strings remaining in the list.	

object.IbDELETSTRING(IIndex As Long) As Long

Parameters

index

Specifies the zero-based index of the string to be deleted. In Windows 95 the this parameter is limited to 16-bit values.

Return Values

The return value is a count of the strings remaining in the list. The return value is -1 if the index parameter specifies an index greater than the number of items in the list.

IbDIR

Adds a list of filenames to a list box and returns the index of the last filename added.

object.IbDIR(IAttributes As Long, sDir As String) As Long

Parameters

IAttributes

Specifies the attributes of the files to be added to the list box. This parameter can be a combination of the following values:

Value	Description
DDL_ARCHIVE	Includes archived files.
DDL_DIRECTORY	Includes subdirectories. Subdirectory names are enclosed in square brackets ([]).
DDL_DRIVES	Includes drives. Drives are listed in the form [-x-], where x is the drive letter.
DDL_EXCLUSIVE	Includes only files with the specified attributes. By default, read-write files are listed even if DDL_READWRITE is not specified.
DDL_HIDDEN	Includes hidden files.
DDL_READONLY	Includes read-only files.
DDL_READWRITE	Includes read-write files with no additional attributes.
DDL_SYSTEM	Includes system files.

sDir

String that specifies the filename to add to the list. If the filename contains wildcards (for example, *.*), all files that match the wildcards and have the attributes specified by the IAttributes parameter are added to the list.

lbFINDSTRING

Return Values

The return value is the zero-based index of the last filename added to the list. If an error occurs, the return value is -1. If there is insufficient space to store the new strings, the return value is -2.

Returns the index of the first string in the list box that matches a given prefix.

object.lbFINDSTRING(lIndexStart As Long, sString As String) As Long

Parameters

lIndexStart

Specifies the zero-based index of the item before the first item to be searched. When the search reaches the bottom of the list box, it continues from the top of the list box back to the item specified by the lIndexStart parameter. If lIndexStart is -1, the entire list box is searched from the beginning. In Windows 95 this parameter is limited to 16-bit values.

sString

Contains the prefix to search for. The search is case independent, so this string can contain any combination of uppercase and lowercase letters.

Return Values

The return value is the index of the matching item, or -1 if the search was unsuccessful.

lbFINDSTRINGEXACT

Returns the index of the string that is equivalent to or prefixed by a given prefix.

object.lbFINDSTRINGEXACT(lIndexStart As Long, sString As String) As Long

Parameters

lIndexStart

Specifies the zero-based index of the item before the first item to be searched. When the search reaches the bottom of the list box, it continues from the top of the list box back to the item specified by the lIndexStart parameter. If lIndexStart is - 1, the entire list box is searched from the beginning. In Windows 95 this parameter is limited to 16-bit values.

sString

String to search for. This string can contain a complete filename, including the extension. The search is not case sensitive, so this string can contain any combination of uppercase and lowercase letters.

Return Values

The return value is the zero-based index of the matching item, or -1 if the search was unsuccessful.

lbGETANCHORINDEX

Returns the index of the item that the mouse last selected.

object.lbGETANCHORINDEX() As Long

Return Values
The return value is the index of the anchor item.
lbGETCARETINDEX Returns the index of the item that has the focus rectangle.

object.lbGETCARETINDEX() As Long

Return Values
The return value is the zero-based index of the list box item that has the focus rectangle. If the list box is a single-selection list box, the return value is the zero-based index of the item that is selected, if any.
lbGETCOUNT Returns the number of items in the list box.

object.lbGETCOUNT() As Long

Return Values
The return value is the number of items in the list box, or -1 if an error occurs.
Remarks
The returned count is one greater than the index value of the last item (the index is zero-based).
lbGETCURSEL Returns the index of the currently selected item.

object.lbGETCURSEL() As Long

Return Values
In a single-selection list box, the return value is the zero-based index of the currently selected item. If there is no selection, the return value is -1.
Remarks
Do not send this message to a multiple-selection list box. To retrieve the indexes of the selected items in a multiple-selection list box, use the lbGETSELITEMS message. To determine whether the item that has the focus rectangle in a multiple selection list box is selected, use the lbGETSEL message.
If sent to a multiple-selection list box, lbGETCURSEL returns the index of the item that has the focus rectangle. If no items are selected, it returns -1.
lbGETHORIZONTAL_EXTENT Returns the scrollable width, in pixels, of a list box.

object.lbGETHORIZONTAL_EXTENT() As Long

Return Values
The return value is the scrollable width, in pixels, of the list box.
lbGETITEMDATA Returns the 32-bit value associated with the given item.

object.lbGETITEMDATA(lIndex As Long) As Long

Parameters
lIndex

Specifies the index of the item. In Windows 95 this parameter is limited to 16-bit values.

Return Values

The return value is the 32-bit value associated with the item, or -1 if an error occurs.

lbGETITEMHEIGHT

Returns the height, in pixels, of an item in a list box.

object.lbGETITEMHEIGHT(lIndex As Long) As Long

Parameters

lIndex

Specifies the zero-based index of the list box item. This index is used only if the list box has the lbsOWNERDRAWVARIABLE style; otherwise, it must be zero. In Windows 95 this parameter is limited to 16-bit values.

Return Values

The return value is the height, in pixels, of each item in the list box. The return value is the height of the item specified by the index parameter if the list box has the lbsOWNERDRAWVARIABLE style. The return value is -1 if an error occurs.

lbGETITEMRECT

Retrieves the client coordinates of the given list box item.

object.lbGETITEMRECT(lIndex As Long, sRect As String) As Long

Parameters

lIndex

Specifies the zero-based index of the item. In Windows 95 this parameter is limited to 16-bit values.

sRect

Points to a RECT structure that will receive the client coordinates for the item in the list box.

Return Values

If an error occurs, the return value is -1.

lbGETLOCALE

Retrieves the locale of the list box. The high-order word contains the country code and the low-order word contains the language identifier.

object.lbGETLOCALE() As Long

Return Values

The return value is a 32-bit value that specifies the current locale of the list box.

lbGETSEL

Returns the selection state of a list box item.

object.lbGETSEL(lIndex As Long) As Boolean

Parameters

lIndex

Specifies the zero-based index of the item. In Windows

95 this parameter is limited to 16-bit values.

Return Values

True if successful or False otherwise.

lbGETSELCOUNT

Returns the number of selected items in a multiple-selection list box.

object.lbGETSELCOUNT() As Long

Return Values

The return value is the count of selected items in the list box. If the list box is a single-selection list box, the return value is -1.

lbGETSELITEMS

Creates an array of the indexes of all selected items in a multiple-selection list box and returns the total number of selected items.

object.lbGETSELITEMS(lCount As Long, lArray As Long) As Long

Parameters

lCount

Specifies the maximum number of selected items whose item numbers are to be placed in the buffer. In Windows 95 this parameter is limited to 16-bit values.

lArray

Points to the first element of a Long array large enough for the number of integers specified by the lCount parameter.

Return Values

The return value is the number of items placed in the buffer. If the list box is a single-selection list box, the return value is -1.

lbGETTEXT

Retrieves the string associated with a given item and the length of the string.

object.lbGETTEXT(lIndex As Long, sString As String) As Long

Parameters

lIndex

Specifies the zero-based index of the string to retrieve. In Windows 95 this parameter is limited to 16-bit values.

sString

String that will receive the value. The string must have sufficient space for the string and a terminating null character. An lbGETTEXTLEN message can be sent before the lbGETTEXT message to retrieve the length, in characters, of the string.

Return Values

The return value is the length of the string, in characters, excluding the terminating null character. If index does not specify a valid index, the return value is -1.

lbGETTEXTLEN Returns the length, in characters, of the string associated with a given item.

`object.lbGETTEXTLEN(lIndex As Long) As Long`

Parameters
lIndex
Specifies the zero-based index of the string. In Windows 95 this parameter is limited to 16-bit values.

Return Values
The return value is the length of the string, in characters, excluding the terminating null character. Under certain conditions, this value may actually be greater than the length of the text. For more information, see the following Remarks section.

lbGETTOPINDEX Returns the index of the first visible item in a list box.

`object.lbGETTOPINDEX() As Long`

Return Values
The return value is the index of the first visible item in the list box.

lbINITSTORAGE Allocates memory for the specified number of items and their associated strings.

`object.lbINITSTORAGE(lItems As Long, lMem As Long) As Long`

Parameters
lItems
Specifies the number of items to add. In Windows 95 this parameter is limited to 16-bit values.

lMem
Specifies the amount of memory, in bytes, to allocate for item strings.

Return Values
The return value is the maximum number of items that the memory object can store before another memory reallocation is needed, if successful. It is -2 if not enough memory is available.

Remarks
Windows 95: This message helps speed up the initialization of list boxes that have a large number of items (more than 100).

Windows NT: This message is not needed on Windows NT.

lbINSERTSTRING Inserts a string at a given index in a list box.

`object.lbINSERTSTRING(lIndex As Long, sString As String) As Long`

Parameters

IIndex

Specifies the zero-based index of the position at which to insert the string. If this parameter is -1, the string is added to the end of the list. In Windows 95 this parameter is limited to 16-bit values.

sString

String to be inserted.

Return Values

The return value is the index of the position at which the string was inserted. If an error occurs, the return value is -1. If there is insufficient space to store the new string, the return value is -2.

lbITEMFROMPOINT

Retrieves the zero-based index of the item nearest the specified point in a list box.

object.lbITEMFROMPOINT(IX As Long, long IY As Long) As Long

Parameters

IX

Specifies the x-coordinate of a point, relative to the upper-left corner of the client area of the list box.

IY

Specifies the y-coordinate of a point, relative to the upper-left corner of the client area of the list box.

Return Values

The return value contains the index of the nearest item in the low-order word. The high-order word is zero if the specified point is in the client area of the list box, or one if it is outside the client area.

lbRESETCONTENT

Removes all items from a list box.

object.lbRESETCONTENT()

lbSELECTSTRING

Selects the first string it finds that matches a given prefix.

object.lbSELECTSTRING(IIndex As Long, sString As String) As Long

Parameters

IIndex

Specifies the zero-based index of the item before the first item to be searched. When the search reaches the bottom of the list box, it continues from the top of the list box back to the item specified by the IIndexStart parameter. If IIndexStart is -1, the entire list box is searched from the beginning. In Windows 95 this parameter is limited to 16-bit values.

sString

String that contains the prefix to search for. The search is case independent, so this string can contain any combination of uppercase and lowercase letters.

lbSELITEMRANGE

Return Values
If the search is successful, the return value is the index of the selected item. If the search is unsuccessful, the return value is -1 and the current selection is not changed.

Selects a given range of items in a list box.

`object.lbSELITEMRANGE(BOOL bSelect, lStart As Long, lStop As Long) As Long`

Parameters
bSelect
Specifies how to set the selection. If the bSelect parameter is True, the string is selected and highlighted; if bSelect is zero, the highlight is removed and the string is no longer selected.

lStart
Specifies the zero-based index of the first item to select.

lStop
Specifies the zero-based index of the last item to select.

Return Values
If an error occurs, the return value is -1.

Remarks
Use this message only with multiple-selection list boxes. This message can select a range only within the first 65,536 items.

lbSELITEMRANGEEX

Selects a given range of items if the index of the first item in the range is less than the index of the last item in the range. Cancels the selection in the range if the index of the first item is greater than the last.

`object.lbSELITEMRANGEEX(lStart As Long, lStop As Long) As Boolean`

Parameters
lStart
Specifies the zero-based index of the first item to select. In Windows 95 this parameter is limited to 16-bit values.

lStop
Specifies the zero-based index of the last item to select.

Return Values
If an error occurs, the return value is -1.

Remarks
If the lStart parameter is less than the lStop parameter, the specified range of items is selected. If lStart is greater than lStop, the selection is removed from the specified range of items.

Use this message only with multiple-selection list boxes. This message can select a range only within the first

65,536 items.

lbSETANCHORINDEX Sets the item that the mouse last selected to a given item.

`object.lbSETANCHORINDEX(IIndex As Long) As Boolean`

Parameters
IIndex
Specifies the index of the new anchor item. In Windows 95 this parameter is limited to 16-bit values.

Return Values
Returns True for success or False for failure.

lbSETCARETINDEX Sets the focus rectangle to a given list box item.

`object.lbSETCARETINDEX(IIndex As Long, bScroll As Boolean) As Boolean`

Parameters
index
Specifies the zero-based index of the list box item that is to receive the focus rectangle. In Windows 95 this parameter is limited to 16-bit values.

bScroll
If this value is False, the item is scrolled until it is fully visible; if it is True, the item is scrolled until it is at least partially visible.

Return Values
If an error occurs, the return value is -1.

lbSETCOLUMNWIDTH Sets the width, in pixels, of all columns in a list box.

`object.lbSETCOLUMNWIDTH(IWidth As Long)`

Parameters
IWidth
Specifies the width, in pixels, of all columns.

lbSETCOUNT Sets the number of items in a list box.

`object.lbSETCOUNT(ICount As Long) As Long`

Parameters
ICount
Specifies the new count of items in the list box. In Windows 95 this parameter is limited to 16-bit values.

Return Values
If an error occurs, the return value is -1. If there is insufficient memory to store the items, the return value is -2.

Remarks
The lbSETCOUNT message is supported only by list boxes created with the lbsNODATA style and not created with the lbsHASSTRINGS style. All other list boxes return

lbSETCURSEL -1.
Selects a given list box item.

`object.lbSETCURSEL(IIndex As Long) As Boolean`

Parameters
IIndex
Specifies the zero-based index of the string that is selected. If the index parameter is -1, the list box is set to have no selection. In Windows 95 this parameter is limited to 16-bit values.

Return Values
If an error occurs, the return value is -1. If the index parameter is -1, the return value is -1 even though no error occurred.

Remarks
Use this message only with single-selection list boxes. You cannot use it to set or remove a selection in a multiple-selection list box.

lbSETHORIZONTAL_EXTENT Sets the scrollable width, in pixels, of a list box.

`object.lbSETHORIZONTAL_EXTENT(IWidth As Long)`

Parameters
IWidth
Specifies the number of pixels by which the list box can be scrolled. In Windows 95 this parameter is limited to 16-bit values.

lbSETITEMDATA Associates a 32-bit value with a list box item.

`object.lbSETITEMDATA(IIndex As Long, IData As Long) As Boolean`

Parameters
IIndex
Specifies the zero-based index of the item. In Windows 95 this parameter is limited to 16-bit values.

IData
Specifies the 32-bit value to be associated with the item.

Return Values
If an error occurs, the return value is -1.

lbSETITEMHEIGHT Sets the height, in pixels, of an item or items in a list box.

`object.lbSETITEMHEIGHT(IIndex As Long, IHeight As Long) As Boolean`

Parameters
IIndex
Specifies the zero-based index of the item in the list box. Use this parameter only if the list box has the `lbsOWNERDRAWVARIABLE` style; otherwise, set it to zero. In Windows 95 this parameter is limited to 16-bit

values.

IHeight

Specifies the height, in pixels, of the item.

Return Values

True for success, False for failure.

lbSETLOCALE

Sets the locale of a list box and returns the previous locale identifier.

object.lbSETLOCALE(ILocale As Long) As Long

Parameters

ILocale

Specifies the locale identifier that the list box will use for sorting when adding text.

Return Values

The return value is the previous locale identifier. If the wLocaleID parameter specifies a locale that is not installed on the system, the return value is -1 and the current list box locale is not changed.

lbSETSEL

Selects an item in a multiple-selection list box.

object.lbSETSEL(BOOL bSelect, IIndex As Long) As Boolean

Parameters

bSelect

Specifies how to set the selection. If the bSelect parameter is True, the string is selected and highlighted; if bSelect is False, the highlight is removed and the string is no longer selected.

IIndex

Specifies the zero-based index of the string to set. If index is -1, the selection is added to or removed from all strings, depending on the value of bSelect.

Return Values

If an error occurs, the return value is -1.

Remarks

Use this message only with multiple-selection list boxes. Sets the tab stops to those specified in a given array.

lbSETTABSTOPS

object.lbSETTABSTOPS(ITabs As Long, ITabArray As Long) As Boolean

Parameters

ITabs

Specifies the number of tab stops in the list box.

ITabArray

Points to the first member of an array of integers containing the tab stops, in dialog box units. The tab stops

must be sorted in ascending order; backward tabs are not allowed.

Return Values

If all the specified tabs are set, the return value is True otherwise, it is False.

Remarks

To respond to the lbSETTABSTOPS method, the list box must have been created with the lbsUSETABSTOPS style. If lTabs parameter is 0, the default tab stop is two dialog box units.

lbSETTOPINDEX

Scrolls the list box so the specified item is at the top of the visible range.

object.lbSETTOPINDEX(lIndex As Long) As Boolean

Parameters

lIndex

Specifies the zero-based index of the item in the list box.

Warning

Windows 95: The lIndex parameter is limited to 16-bit values. This means list boxes cannot contain more than 32,767 items. Although the number of items is restricted, the total size in bytes of the items in a listbox is limited only by available memory.

Return Values

True for success, False for Failure.

Remarks

The system scrolls the list box contents so that either the specified item appears at the top of the list box or the maximum scroll range has been reached.

Refresh

Standard Method

ReleaseFont

Releases the font established by FontSelect

RemoveItem

Standard Method

Events (List Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Event Description	
Click	Standard Event	
Command	LBN_DBLCLK	The user double-clicks an item in the list box.
	LBN_ERRSPACE	The list box cannot allocate enough memory to fulfill a request.
	LBN_KILLFOCUS	The list box loses the keyboard focus.
	LBN_SELCANCEL	The user cancels the selection of an item in the list box.
	LBN_SELCHANGE	The selection in a list box is about to change.
	LBN_SETFOCUS	The list box receives the keyboard focus.
DblClick	Standard Event	
KeyDown	Standard Event	
KeyPress	Standard Event	
KeyUp	Standard Event	
MouseDown	Standard Event	
MouseMove	Standard Event	
MouseUp	Standard Event	
Resize	Standard Event	

CALL BACKS

Callback Name	Callback Description
DrawItem	<p>DrawItem is a prototype for a call back procedure. It does not have to be named DrawItem, but it must have two parameters with the correct type and be a SUB (not a function returning a value).</p> <p>You place the address of this routine into the ItemDrawCB property and set the cbOWNERDRAW property to True and this routine will be called to perform the painting of the item.</p>

Sub DrawItem (objItem As Object, dw As DRAWITEMSTRUCT)

objItem the control being painted. You can access the value of the properties from this Object.

dw is a draw item structure as used in all owner draw controls in Windows.

MeasureItem

MeasureItem is a prototype for a call back procedure. It does not have to be named MeasureItem, but it must have two parameters with the correct type and be a SUB (not a function returning a value).

You place the address of this routine into the ItemMeasureCB property and set the lbsOWNERDRAW property to True and this routine will be called to perform the sizing of an owner draw item.

Sub MeasureItem (objItem As Object, dw As MEASUREITEMSTRUCT)

objItem the control being painted. You can access the value of the properties from this Object.

dw is a draw item structure as used in all owner draw controls in Windows.

WindowProc

A callback procedure used to respond to or process any Windows message. The list box control is already subclassed for you eliminating the need to write or use a subclassing control to get to any Windows' message.

Sub WinProc(objItem As Object, msg As Long, wp As Long, lp As Long, _ bDone As Long)

objItem the control this message belongs to. You can access the value of the properties from this Object

msg the message value, WM_...

wp the WPARAM value for the message.

lp the LPARAM value for the message.

bDone a flag that can be set to skip processing this message, also indicates before or after processing on entry.

Notice that the WinProc contains essentially the same information as any Windows message. The typical hWnd parameter has been replaced with the control and the bDone parameter has been added. It is possible and may be necessary to do some creative programming to get the value that is represented by the lp parameter as this value could be a direct long value or a pointer to almost any type of data. bDone is set to FALSE on entry to the WinProc Sub when firing the event for before default processing. It will be TRUE when the routine is processing the message after default processing. When bDone is FALSE, it may be set to TRUE before exiting the routine to skip

default processing. When this is done, the “after” event will not be fired.

List View Constituent

Properties

Methods

Events



Class: bbbListView

Sample-ListView\bbbListView.vbg

Overview

Much like the standard ListView, the BeCubed ListView constituent provides the ability to add and remove items from a list. Items in the list may contain bitmap images that are stored in an associated image list control. The BeCubed ListView constituent is a good example of the dramatic difference that can exist between the functionality of a standard Visual Basic control and a BeCubed constituent. The BeCubed version provides owner-draw capabilities, the ability to determine how many list items are visible, coordinates of list items, a callback function that lets you determine how you would like the list items sorted and access to all Windows messages.

The standard ListView control uses the ListItem object and the ListItems collection to add items to the ListView. These are components of the standard ListView control but not the base window class. For more on window classes, see the **Creating ActiveX Controls using the BeCubed Constituent Wizard** section earlier in this manual.

To add an item to the standard ListView control, you use the following syntax:

```
ListView1.ListItems.Add(Object, Index, Key, Text, Icon, SmallIcon)
```

The base window class has neither a list item object or a list items collection (collection where all of the list items are stored). Since this is the case, if we wish to preserve existing source code, we must recreate them. The good news is that the samples provided with the constituents have already done this for you. Let's examine the implications that this has on the development of ActiveX controls.

At first glance, you may think that the lack of the ListItems collection and ListItem object in the base class equate to a loss of functionality. In reality, the exact opposite is true. The key to understanding ActiveX development is to realize that a control is built from the ground up and is based on a specific set of requirements. If you start the development of a control locked into the preset functionality of another control, you are prevented from implementing the features that you want in the way that you want to. In this case, you are not really writing an ActiveX control, you are simply using someone else's. Additionally, your control may not require the additional overhead that the list item and other such objects and features require. This additional overhead is confusing to the users of your control and increases the size of distribution disks.

Consider the example presented earlier in the **Creating ActiveX Controls using the BeCubed Constituent Wizard** section. We have a command button, a list box and a text box on our user control. When the command button is pressed, the item in the text box is added to the list box. When an item is added to the list box, we in turn want it added to a database. Most importantly, we want to preserve the AddItem syntax of the standard list box so that we do not have to change our existing code. If we are locked into the functionality of the standard AddItem method, the addition of items to the database becomes more convoluted.

Note: In this situation, you would probably want to add a property denoting the location of the database as well as a property indicating whether or not you want new items added to the database.

Open the group ListView sample project provided with the constituents. Make sure that you have opened the group project. This sample contains 2 projects. The first is an ActiveX control that implements the ListView constituent. The second is a sample that exercises the features of the constituent.

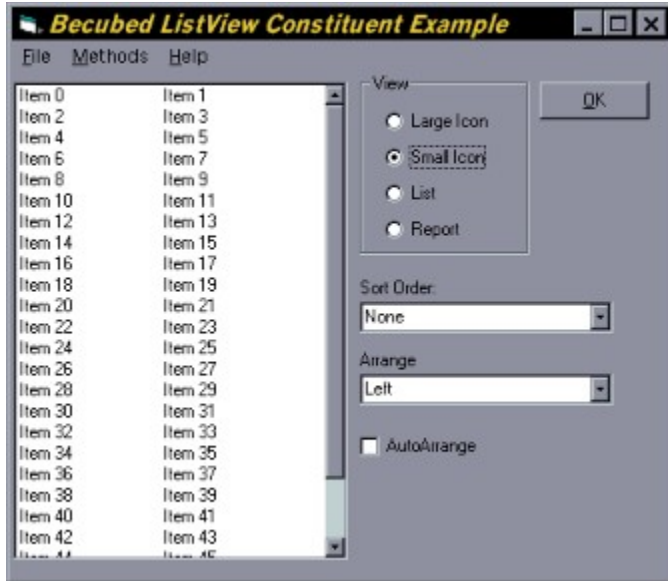


Figure 13: The ListView Constituent Control Example Program

Take a look at the components of the ActiveX project. Notice that under the Classes section, 2 classes exist: B3ListItem and B3ListItems. B3ListItem implements a list item object and B3ListItems a list items collection (the collection where the individual B3ListItems are stored. If you take a look at the B3ListItem object, you will notice the it contains the same methods that the standard ListView control implements for its ListItem object.

Take a look at the AddItem method for the B3ListItem class. In this sample, the AddItem method simply calls the lvmInsertItem method of the ListView constituent. Let's take a look at how this works.

The user control contains a property called ListItems. This is an instance of our B3ListItems class object. When a user addresses the ActiveX control with MyControl.ListItems.Add(), they are actually addressing the ListItems property of the user control. This ListItems property (our B3ListItems class instance) is itself an object with a method called Add. This method adds a new item to the ListView constituent.

This may sound a bit convoluted at first, but consider the possibilities. In our previous example we wanted to automatically add each new list item to a database. To accomplish this in our ListView, we simply add the database code to the B3ListItems object's Add method. This eliminates the need for the user to write additional database code, while preserving existing syntax.

Another important point to consider is functionality overkill. Imagine that all you wanted your ActiveX control to do is to add items to a list box and database. If you were forced into using the standard list item object and list items collection, your control would contain a large number of unwanted properties and methods. This not only makes it more confusing for people to understand the functionality of your control, it adds considerable overhead during distribution (comctl32.ocx, for example is 597,856 bytes compared to 50,688 for the ListView constituent.)

If we view the construction of ActiveX components in this light, we approach control development by asking what functionality we want our control to have and then combining the components that we need to achieve that functionality. It is definitely a different paradigm, but once understood, the quality of your controls will improve dramatically.

The following section outlines the extended features of the BeCubed ListView constituent control.

Features

This section covers the extended features of the ListView control. For ease of use, we only cover items that are not available in the standard ListView control.

Draw directly into the ListView control at run time without using an image list. (owner-draw).

One of the most interesting features that the ListView constituent provides is owner-draw capabilities. Owner-draw allows you to draw directly into the items of a ListView control at run time. Images drawn in the ListView are automatically preserved by the ListView constituent. If you were to do this with a standard ListView, the drawing would disappear once the list item was scrolled out of (and back into) view. All management of the drawing is handled by the ListView constituent.

To utilize owner-draw capabilities, set the `lvsOWNERDRAWFIXED` property to `True`. This enables the list to repaint items automatically. Owner-draw capabilities are available in report view only.

Retrieve the coordinates of a list item.

In order to draw directly into a list item, you must know exactly where that item is located. The `lvmGETITEMRECT` method retrieves the rectangle of a specific list item. This method takes the index of the list item whose coordinates you are seeking.

Insert a new column into the ListView control.

The `lvmINSERTCOLUMN` method inserts a new column into the ListView control. Use `lvmGETCOLUMN` to retrieve attributes about a column.

See the Constituent Control Reference later in this manual for more information on the parameters to this method.

Remove a column from the ListView control.

The `lvmDELETECOLUMN` method removes an existing column from the ListView control.

See the Constituent Control Reference in the back of this manual for more information on the parameters to this method.

Retrieve the width of a column in the ListView.

The `lvmGETCOLUMNWIDTH` method allows you to retrieve the width of a column when in report or ListView. It takes the column number as a parameter and returns the width of a column.

```
lWidth = BBBListView1.lvmGETCOLUMNWIDTH (1)
```

Determine the number of items that are currently visible to the user.

It may be important to determine how many items exist within the visible area of the ListView control at any given time. If the user resizes a form, and you in turn resize your control, the number of visible list items may change. If you need to determine how many items are displayed in the visible area, use the `lvmGETCOUNTPERPAGE` method. This method requires no parameters and returns the count of the items that can be displayed:

```
iCount = BBBListView1.lvmGETCOUNTPERPAGE ()
```

Determine the number of items that are currently selected.

If extended selection is in force, the user may select more than one list item. The `lvmGETSELECTEDCOUNT` method returns the number of items that are currently selected in the ListView.

```
lRet = BBBListView1.lvmGETSELECTEDCOUNT ()
```

Sort the items in the ListView control with a user-defined sorting function.

One of the features of the ListView constituent is the ability to sort the list items in any way you choose. This is accomplished using the `lvmSORTITEMS` method. Your application supplies the function that does the sorting. The function is passed to the ListView using the `AddressOf` operator. As items are sorted, the ListView passes pairs of items to your function for comparison. This is implemented as a callback. Your application receives a callback from the ListView control as the pairs of items are sorted.

See the Constituent Control Reference in the back of this manual for more information on the parameters to this method.

Access the edit control within a ListView.

In the BeCubed ListView control, you have direct access to the edit control that is displayed when editing items. When a ListView control is in edit mode, a rectangle appears around the list item being edited. At this point, the user may change the text associated with that entry. To initiate an edit in the constituent, you can also use the `lvmEDITLABEL` method. When a label edit begins, the ListView control receives a `BeforeLabelEdit` event. Once

the editing has completed, an AfterLabelEdit event occurs.

Scroll the ListView to ensure that a specific item is visible.

Scrolling the ListView to display a specific item is accomplished by using the lvmENSUREVISIBLE method. This method takes the index of the list item you want to ensure is visible, and a Boolean indicating whether or not the ListView should scroll if the item is already partially visible. The following example ensures that the tenth item in the ListView is visible. If the item is already partially visible, the list scrolls to fully display the item. The return value is true if the item was successfully scrolled into view.

```
bRet = BBBListView1.lvmENSUREVISIBLE (10, TRUE)
```

Scroll the ListView to a specific position.

The lvmSCROLL method can be used to scroll the list of items. This method takes 2 parameters that determine the amount of horizontal and vertical scrolling respectively.

See the Constituent Control Reference in the back of this manual for more information on the parameters to this method.

Locate a specific item within the ListView.

The lvmFINDITEM method allows you to search for a specific item within the ListView. The method allows you to specify what to search for, as well as where to begin the search (-1 indicates to begin at the top of the list). You can also specify the direction in which to search and whether to wrap around to the top of the list if the item is not found.

You may also specify whether you want to match the search string exactly, or want to locate the first list item that begins with the search string (For example, "Miss", locates "Mississippi").

The following code searches a bbbListView control for the string sSearchString. In order for the search to be successful, the string in the ListView must *exactly* match sSearchString.

```
BBBListView1.lvmFINDITEM(-1, LVFI_STRING, sSearchString, 0, 0, 0, 0)
```

-1 Start at top of list
LVFI_STRING Search the list for an exact match
sSearchString String to search for

Additionally, each item of the ListView control has an lParam member you can use to store numbers, pointers to objects, etc. The lParam member of the ListView can also be searched for a specific entry. In the next example, lParamItem is the item that you are searching for:

```
BBBListView1.lvmFINDITEM(-1, LVFI_PARAM, lParamItem , 0, 0, 0, 0)
```

-1 Start at top of list
LVFI_PARAM Search the lParam member of the bbbListView
lParamItem Item to search for

See the Constituent Control Reference later in this manual for more information about this method.

Move an item to a new location within the ListView.

The lvmSETITEMPOSITION and lvmSETITEMPOSITION32 methods allow you to move items to different locations within the ListView. This method takes three parameters, the item to be moved and the X and Y coordinates of the new location.

Retrieve an incremental search string of characters entered by the user.

The ListView constituent maintains an incremental search string of characters entered by a user. The incremental search string is the character sequence the user types while the ListView has the input focus.

For example, imagine that the following items are in your ListView:

James
Joan
Joanie

When the user types “J”, the list scrolls to the first entry in the ListView that begins with J (James). At this point, the incremental search string of the ListView control contains the letter “J”. If the user quickly types the letter “o”, the incremental search string is updated to contain “Jo” and the list is scrolled to Joan. If the user pauses too long, the incremental search string times out and is reset. Each time the a new character is typed (within the timeout period), the incremental string is updated and the ListView scrolls to the item that most closely matches the string.

To retrieve the incremental search string, use the `lvmGETISEARCHSTRING` method. This method requires no parameters and returns either the search string, or NULL if the control is not in incremental search mode (the ListView does not have the input focus).

Determine the spacing between items in the ListView.

Use the `lvmGETITEMSPACING` method to determine much space exists between list items. The method takes one parameter that asks whether you want spacing information for icon or small icon view. The following code fragment retrieves the spacing in small icon view:

```
Dim lSpacing As Long
Dim fSmall As Boolean

fSmall = True
lSpacing = BBListView1.lvmGETITEMSPACING(fSmall)
```

Determine the width of a string using the currently selected font.

The `lvmGETSTRINGWIDTH` method returns the width (in pixels) of the text of an item in the ListView. This method is particularly useful for determining the width of a column. You can use the return from this method to call `lvmSETCOLUMNWIDTH` (remember to pad the column width a bit so that the string is not truncated).

```
lWidth = BBListView1.lvmGETSTRINGWIDTH(sText)
```

Retrieve the coordinates of all items in the ListView control.

The `lvmGETVIEWRECT` method returns the bounding rectangle of all items in the ListView control. This method can be used in icon or small icon view.

Prepare to add a large number of items to the ListView.

If you are planning to add a large number of items to the ListView control, use `lvmSETITEMCOUNT`. This method takes 1 parameter, the number of items that the ListView will ultimately contain.

Properties (List View Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
Arrange		
BackColor	Standard	Property
BorderStyle	Standard	Property
Container	Standard	Property
DragIcon	Standard	Property
DragMode	Standard	Property
Enabled	Standard	Property
Font	Standard	Property
ForeColor	Standard	Property
Height	Standard	Property
HelpContextID	Standard	Property
hWnd	Standard	Property
Index	Standard	Property
Left	Standard	Property
lvsALIGNLEFT		Specifies that items are left-aligned in icon and small icon view.
lvsALIGNTOP		Specifies that items are aligned with the top of the list view control in icon and small icon view.
lvsAUTOARRANGE		Specifies that icons are automatically kept arranged in icon and small icon view.
lvsBUTTON		Specifies that item icons look like buttons in icon view.
lvsEDITLABELS		Allows item text to be edited in place. The parent window must process the LVN_ENDLABELEDIT notification message.
lvsICON		Specifies icon view.
lvsLIST		Specifies list view.
lvsNOCOLUMN_ HEADER		Specifies that a column header is not displayed in report view. By default, columns have headers in report view.
lvsNOLABELWRAP		Displays item text on a single line in icon view. By default, item text may wrap in icon view.
lvsNOSCROLL		Disables scrolling. All items must be within the client area.
lvsNOSORTHEADER		Specifies that column headers do not work like buttons. This style is useful if clicking a column header in report view does not carry out an action, such as sorting.
lvsOWNERDRAW_ FIXED		Enables the owner window to paint items in report view. The list view control sends a WM_DRAWITEM message to paint each item; it does not send separate messages for each subitem. The itemData member of the DRAWITEMSTRUCT structure contains the item data for the specified list view item.
lvsREPORT		Specifies report view. When using the lvsREPORT style with a List View control, the first column is always left-aligned. You can not use LVCFMT_RIGHT to change this alignment.
lvsSHAREIMAGELISTS		Specifies that the control does not take ownership of the image lists assigned to it; that is, it does not destroy the image lists when it is destroyed. This style enables the same image lists to be used with multiple list view

	controls.
lvsSHOWSELALWAYS	Always show the selection, if any, even if the control does not have the focus.
lvsSINGLESEL	Allows only one item at a time to be selected. By default, multiple items may be selected.
lvsSMALLICON	Specifies small icon view.
lvsSORTASCENDING	Sorts items based on item text in ascending order.
lvsSORTDESCENDING	Sorts items based on item text in descending order.
MouseIcon	Standard Property
MousePointer	Standard Property
Name	Standard Property
Object	Standard Property
Parent	Standard Property
SelectedItem	Standard Property
TabIndex	Standard Property
TabStop	Standard Property
Tag	Standard Property
Top	Standard Property
Visible	Standard Property
WhatsThisHelpID	Standard Property
Width	Standard Property

Methods (List View Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Method Description	
Drag lvmARRANGE	Standard Method Arranges items in icon view. object.lvmARRANGE(code As Long) Parameters: code Specifies the alignment, which can be one of the following values: LVA_ALIGNLEFT Aligns items along the left edge of the window. LVA_ALIGNTOP Aligns items along the top edge of the window. LVA_DEFAULT Aligns items according to the list view control's current alignment styles (the default value). LVA_SNAPTOGRID Snaps all icons to the nearest grid position. Return Value (Boolean) Returns TRUE if successful or FALSE otherwise. Creates a drag image list for the specified item. object.lvmCREATEDRAGIMAGE(item As Long, x As Long, y As Long) Parameters item Index of the item. x, y upper-left corner of the image, in view coordinates. Return Value (Long) Returns the handle to the drag image list if successful or NULL otherwise. Returns True if successful or False if the method failed. object.lvmDELETEALLITEMS() Parameters N/A Return Value (Boolean) Returns TRUE if successful or FALSE otherwise. Removes a column from a list view control. Returns True	
lvmCREATEDRAG_IMAGE		
lvmDELETEALLITEMS		
lvmDELETECOLUMN		

if successful or False if the method failed.

object.lvmDELETETECOLUMN(Col As Long)

Parameters

Col

Index of the column to delete.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

lvmDELETEITEM

removes an item from a list view control. Returns True if successful or False if the method failed.

object.lvmDELETEITEM (item As Long)

Parameters

item

Index of the list view item to delete.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

lvmEDITLABEL

Returns the hWnd of the edit control used to edit the label. This method implicitly selects and focuses the specified item.

object.lvmEDITLABEL(iItem As Long)

Parameters

iItem

Index of the list view item. To cancel editing, set iItem to -1.

Return Value (Long)

Returns the handle of the edit control that is used to edit the item text if successful or NULL otherwise.

Remarks

When the user completes or cancels editing, the edit control is destroyed and the handle is no longer valid. You can safely subclass the edit control, but you should not destroy it.

The control must have the focus before you send this message to the control. Focus can be set using the SetFocus function.

lvmENSUREVISIBLE

ensures that a list view item is entirely or at least partially visible, scrolling the list view control if necessary.

object.lvmENSUREVISIBLE(index As Long, fPartialOK As Boolean)

Parameters

index

Index of the list view item.

fPartialOK

Value specifying whether the item must be entirely visible. If this parameter is TRUE, no scrolling occurs if the item is at least partially visible.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

lvmFINDITEM

searches for a list view item with the specified characteristics.

object.lvmFINDITEM(Start As Long, flags As Long, ByVal psz As String, lParam As Long, x As Long, Y As Long, vkDirection As Long)

Parameters

iStart

Index of the item to begin the search with or -1 to start from the beginning. The specified item is itself excluded from the search.

flags

Type of search to perform. This member can be one or more of the following values:

LVFI_PARAM Searches based on the lParam member. The lParam member of the matching item must match lParam. If this flag is specified, all other flags are ignored.

LVFI_PARTIAL Matches if the item text begins with the string pointed to by the psz member. This value implies use of LVFI_STRING.

LVFI_STRING Searches based on the item text. Unless additional values are specified, the item text of the matching item must exactly match the string pointed to by the psz member.

LVFI_WRAP Continues the search at the beginning if no match is found.

LVFI_NEARESTXY Finds the item nearest the specified position in the specified direction.

psz

Pointer to a null-terminated string to compare with the

item text if flags specifies LVFI_STRING or LVFI_PARTIAL.

lParam

Value to compare with the lParam member of a list view item structure if the flags Parameter specifies LVFI_PARAM.

x, y

specifies the starting position to search from. These parameters are used only if LVFI_NEARESTXY is specified.

vkDirection

Direction to search in. This member is used only if LVFI_NEARESTXY is specified. If this member used, it specifies the virtual-key code of an arrow key.

Return Value (Long)

Returns the index of the item if successful or -1 otherwise. Returns the BackColor of the ListView control.

lvmGETBKCOLOR

object.lvmGETBKCOLOR()

Parameters

N/A

Return Value (Long)

Returns the background color of the list view control.

lvmGETCALLBACK_MASK

Retrieves the callback mask for a list view control.

Returns the call back mask. (See lvmSETCALLBACKMASK)

object.lvmGETCALLBACKMASK()

Parameters

N/A

Return Value (Long)

Returns the callback mask.

lvmGETCOLUMN

Retrieves the attributes of a list view control's column.

object.lvmGETCOLUMN(Col As Long, mask As Long, fmt As Long, cx As Long, pszText As String, SubItem As Long)

Parameters

Col

Index of the column.

mask

Specifies which members of this structure contain valid information. This member can be zero, or one or more of the following values:

LVCF_FMT

The fmt parameter is

valid.

LVCF_SUBITEM The SubItem parameter is valid.

LVCF_TEXT The pszText parameter is valid.

LVCF_WIDTH The cx parameter is valid.

fmt

Specifies the alignment of the column heading and the subitem text in the column. This member can be one of the following values:

LVCFMT_CENTER Text is centered.

LVCFMT_LEFT Text is left-aligned.

LVCFMT_RIGHT Text is right-aligned.

Note: The leftmost column in a list view control must be left aligned.

cx

Specifies the width, in pixels, of the column.

pszText

String that contains the column heading.

SubItem

Specifies the index of the subitem associated with column.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

lvmGETCOLUMN_
WIDTH

Retrieves the width of a column in report or list view.

object.lvmGETCOLUMNWIDTH (Col)

Parameters

Col

Index of the column. This parameter is ignored in list view.

Return Value (Long)

Returns the column width if successful or zero otherwise.

lvmGETCOUNTPER_
PAGE

Calculates the number of items that can fit vertically in the visible area of a list view control when in list or report view. Only fully visible items are counted.

Returns the number of fully-visible items if successful.

object.lvmGETCOUNTPERPAGE ()

Parameters

N/A

Return Value (Long)

Returns the number of fully-visible items if successful. If the current view is icon or small icon view, the return value is the total number of items in the list view control.

lvmGETEDITCONTROL

Retrieves the handle to the edit control being used to edit a list view item's text. If no label is being edited, the return value is NULL. The edit control is not created until after the LVN_BEGINLBELEDIT notification message is sent.

object.lvmGETEDITCONTROL ()

Parameters

N/A

Return Value (Long)

Returns the handle to the edit control if successful or NULL otherwise.

Remarks

If no label is being edited, the return value is NULL. The edit control is not created until after the LVN_BEGINLBELEDIT notification message is sent. When the user completes or cancels editing, the edit control is destroyed and the handle is no longer valid. You can safely subclass the edit control, but you should not destroy it. To cancel editing, you can send the list view control a WM_CANCELMODE message. The list view item being edited is the currently focused item ³/₄ that is, the item in the focused state. To find an item based on its state, use the lvmGETNEXTITEM message.

See Also

LVN_BEGINLBELEDIT, lvmGETEDITCONTROL, lvmGETNEXTITEM, WM_CANCELMODE

lvmGETIMAGELIST

Retrieves the handle to an image list used for drawing list view items.

object.lvmGETIMAGELIST(ImageList)

Parameters

ImageList

Image list to retrieve. This parameter can be one of the following values:

LVSIL_NORMAL	Image list with large icons
LVSIL_SMALL	Image list with small icons
LVSIL_STATE	Image list with state

images

lvmGETISEARCH_STRING	<p>Return Value (LONG) Returns the handle of the specified image list if successful or NULL otherwise. Retrieves the incremental search string of a list-view control.</p>
	<p>object.lvmGETISEARCHSTRING()</p>
	<p>Parameters N/A</p>
	<p>Return Value (String) The incremental search string or NULL if the list-view control is not in incremental search mode.</p>
	<p>Remarks The incremental search string is the character sequence that the user types while the list view has the input focus. Each time the user types a character, the system appends the character to the search string and then searches for a matching item. If the system finds a match, it selects the item and, if necessary, scrolls it into view. A timeout period is associated with each character that the user types. If the timeout period elapses before the user types another character, the incremental search string is reset.</p>
lvmGETITEM	<p>retrieves some or all of a list view item's attributes.</p>
	<p>object.lvmGETITEM (mask As Long, item As Long, SubItem As Long, state As Long, stateMask As Long, pszText As String, Image As Long, lParam As Long)</p>
	<p>Parameters mask A set of bit flags that specify attributes of this data structure or of an operation that is using this structure. The following bit flags specify the members of the LV_ITEM structure that contain valid data or need to be filled in. One or more of these bit flags may be set:</p>
	<p>LVIF_TEXT The pszText member is valid or needs to be filled in.</p>
	<p>LVIF_IMAGE The iImage member is valid or needs to be filled in.</p>
	<p>LVIF_PARAM The lParam member is valid or needs to be filled in.</p>
	<p>LVIF_STATE The state member is valid or needs to be filled in.</p>

item

Specifies the zero-based index of the item to which this method refers.

SubItem

Specifies the one-based index of the subitem to which this structure refers, or zero if this structure refers to an item rather than a subitem.

state

Specifies the current state of the item if the item's state is being retrieved, or the new state if the item's state is being set. The stateMask member specifies the bits of the state member that are valid. This member can be any valid combination of state values.

stateMask

Specifies the bits of the state member that are valid.

pszText

String that contains the item text if the method specifies item attributes. If this member is the LPSTR_TEXTCALLBACK value, the item is a callback item. Do not set the pszText member to LPSTR_TEXTCALLBACK if the list view control has lvSORTASCENDING or lvSORTDESCENDING style. If the structure is receiving item attributes, this member is the pointer to the buffer that receives the item text.

Image

Index of the list view item's icon in the icon and small icon image lists. If this member is the I_IMAGECALLBACK value, the parent window is responsible for storing the index. In this case, the list view control sends the parent an LVN_GETDISPINFO notification message to get the index when it needs to display the image.

IParam

A 32-bit value to associate with the item. If you use the LVM_SORTITEMS message, the list view control passes this value to the application-defined comparison function. You can also use the LVM_FINDITEM message to search a list view control for an item with a specified IParam value.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

Remarks

The LV_ITEM structure, which in this method is broken into its respective members, is used with a number of messages, including lvmGETITEM, lvmSETITEM,

lvmsINSERTITEM, and lvmsDELETEITEM.

See Also

[lvmsDELETEITEM](#), [lvmsINSERTITEM](#), [lvmsSETITEM](#)

lvmsGETITEMCOUNT

Retrieves the number of items in a list view control.

object.lvmsGETITEMCOUNT()

Parameters

N/A

Return Value (Long)

Returns the number of items.

lvmsGETITEMPOSITION

Retrieves the position of a list view item.

object.lvmsGETITEMPOSITION(index As Long, x As Long, y As Long)

Parameters

hWnd

Handle to the list view control.

index

Index of the list view item.

x, y

Receives the position of the item's upper-left corner, in view coordinates.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

lvmsGETITEMRECT

retrieves the bounding rectangle for all or part of an item in the current view.

object.lvmsGETITEMRECT(index As Long, Top As Long, Left As Long, Right As Long, Bottom As Long)

Parameters

index

Index of the list view item.

left, top, right, bottom

Receives the bounding rectangle.

code

Portion of the list view item for which to retrieve the bounding rectangle. This parameter can be one of the following values:

LVIR_BOUNDS

Returns the bounding rectangle of the entire item, including the icon and label.

LVIR_ICON

Returns the bounding rectangle of the icon or

		small icon.
	LVIR_LABEL	Returns the bounding rectangle of the item text.
	LVIR_SELECTBOUNDS	Returns the union of the LVIR_ICON and LVIR_LABEL rectangles, but excludes columns in details view.
		Return Value (Boolean) Returns TRUE if successful or FALSE otherwise.
lvmGETITEMSPACING		Determines the spacing between items in a list view control.
		object.lvmGETITEMSPACING(fSmall As Boolean)
		Parameters fSmall View to retrieve the item spacing for. This parameter is TRUE for small icon view, or FALSE for icon view.
		Return Value (Long) Returns the amount of spacing between items.
lvmGETITEMSTATE		Retrieves the state of a list view item.
		object.lvmGETITEMSTATE(index As Long, mask As Long)
		Parameters index Index of the list view item.
		mask Mask that specifies which of the item's state flags to return.
		Return Value (Long) Returns the item's state flags.
lvmGETITEMTEXT		Retrieves the text of a list view item or subitem.
		object.lvmGETITEMTEXT (item As Long, SubItem As Long)
		Parameters iItem Index of the list view item.
		iSubItem Index of the subitem, or zero to retrieve the item label.
		Return Value (String) The item or subitem text.
lvmGETNEXTITEM		searches for a list view item that has the specified properties and bears the specified relationship to a

specified item.

object.lvmGETNEXTITEM (Start As Long, flags As Long, state As Long)

Parameters

Start

Index of the item to begin the search with, or -1 to find the first item that matches the specified flags. The specified item itself is excluded from the search.

flags

Geometric relation of the requested item to the specified item and, if specified, the state of the requested item.

The geometric relation can be one of the following values:

LVNI_ABOVE	Searches for an item that is above the specified item.
LVNI_ALL	Searches for a subsequent item by index (the default value).
LVNI_BELOW	Searches for an item that is below the specified item.
LVNI_TOLEFT	Searches for an item to the left of the specified item.
LVNI_TORIGHT	Searches for an item to the right of the specified item.

The state can be zero, or it can be one or more of the following values:

LVNI_CUT	The item has the LVIS_CUT state flag set.
LVNI_DROPHILITED	The item has the LVIS_DROPHILITED state flag set.
LVNI_FOCUSED	The item has the LVIS_FOCUSED state flag set.
LVNI_SELECTED	The item has the LVIS_SELECTED state flag set.

If an item does not have all of the specified state flags set, the search continues with the next item.

Return Value (Long)

Returns the index of the next item if successful or -1

otherwise.
lvmGETORIGIN Retrieves the current view origin for a list view control.

object.lvmGETORIGIN (x As Long, y As Long)

Parameters
x, y
Receives the view origin.

Return Value (Boolean)
Returns TRUE if successful or FALSE if the current view is list or report view.

lvmGETSELECTED_COUNT Determines the number of selected items in a list view control.

object.lvmGETSELECTEDCOUNT ()

Parameters
N/A

Return Value (Long)
Returns the number of selected items.

lvmGETSTRINGWIDTH Determines the width of a specified string, using the specified list view control's current font.

object.lvmGETSTRINGWIDTH (psz As String)

Parameters
psz
String.

Return Value (Long)
Returns the string width if successful or zero otherwise.

Remarks
This method returns the exact width, in pixels, of the specified string. If you use the returned string width as the column width in a call to the lvmSETCOLUMNWIDTH method, the string will be truncated. To get the column width that can contain the string without truncating it, you must add padding to the returned string width.

See Also
lvmSETCOLUMNWIDTH

lvmGETTEXTBK_COLOR Retrieves the text background color of a list view control.

object.lvmGETTEXTBKCOLOR ()

Parameters
N/A

Return Value (Long)
Returns the background color of the text.

lvmGETTEXT_COLOR Retrieves the text color of a list view control.

object.lvmGETTEXTCOLOR ()

Parameters

Return Values

Returns the text color.

lvmGETTOPINDEX

Retrieves the index of the topmost visible item when in list or report view.

object.lvmGETTOPINDEX ()

Parameters

N/A

Return Value (Long)

Returns the index of the item if successful or zero if the list view control is in icon or small icon view.

lvmGETVIEWRECT

Retrieves the bounding rectangle of all items in the list view control. The list view must be in icon or small icon view.

object.lvmGETVIEWRECT (left As Long, top As Long, right As Long, bottom As Long)

Parameters

left, top, right, bottom

receives the bounding rectangle. All coordinates are relative to the visible area of the list view control.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

lvmHITTEST

Determines which list view item, if any, is at a specified position.

object.lvmHITTEST (x As Long, y As Long, flags As Long, Item As Long)

Parameters

x, y

Position to hit test, in client coordinates.

flags

Variable that receives information about the results of a hit test. This member can be one or more of the following values:

LVHT_ABOVE

The position is above the client area of the control.

LVHT_BELOW

The position is below the client area of the control.

LVHT_NOWHERE

The position is inside the list view control's client window, but it is not over a list item.

LVHT_ONITEMICON	The position is over a list view item's icon.
LVHT_ONITEMLABEL	The position is over a list view item's text.
LVHT_ONITEMSTATEICON	The position is over the state image of a list view item.
LVHT_TOLEFT	The position is to the left of the list view control's client area.
LVHT_TORIGHT	The position is to the right of the list view control's client area.

Remarks

You can use LVHT_ABOVE, LVHT_BELOW, LVHT_TOLEFT, and LVHT_TORIGHT to determine whether to scroll the contents of a list view control. Two of these values may be combined $\frac{3}{4}$ for example, if the position is above and to the left of the client area. You can test for LVHT_ONITEM to determine whether a specified position is over a list view item. This value is a bitwise-OR operation on LVHT_ONITEMICON, LVHT_ONITEMLABEL, and LVHT_ONITEMSTATEICON.

Item

Receives the index of the matching item.

Return Value (Long)

Returns the index of the item at the specified position, if any, or -1 otherwise.

lvmsINSERTCOLUMN

Inserts a new column in a list view control.

object.lvmsINSERTCOLUMN(Col As Long, mask As Long, fmt As Long, cx As Long, pszText As String, SubItem As Long)

Parameters

See lvmsGETCOLUMN

Return Value (Long)

Returns the index of the new column if successful or -1 otherwise.

See Also

lvmsGETCOLUMN

lvmsINSERTITEM

Inserts a new item in a list view control.

object.lvmsINSERTITEM (mask As Long, item As Long, SubItem As Long, state As Long, stateMask As Long, pszText As String, Image As Long, lParam As Long)

Parameters
see lvmGETITEM

Return Value (Long)
Returns the index of the new item if successful or -1 otherwise.

Remarks
If a list view control has either the LVS_SORTASCENDING or LVS_SORTDESCENDING window style, an LVM_INSERTITEM message will fail if you try to insert an item that has LPSTR_TEXTCALLBACK as the pszText member of its LV_ITEM structure.

lvmREDRAWITEMS Forces a list view control to redraw a range of items.

object.lvmREDRAWITEMS (First As Long, Last As Long)

Parameters
First
Index of the first item to redraw.

Last
Index of the last item to redraw.

Return Value (Boolean)
Returns TRUE if successful or FALSE otherwise.

Remarks
The specified items are not actually redrawn until the list view window receives a WM_PAINT message to repaint. To repaint immediately, call the UpdateWindow function after using this macro.

lvmSCROLL Scrolls the content of a list view control.

object.lvmSCROLL(dx As Long, dy As Long)

Parameters
dx
Integer value that specifies the amount of horizontal scrolling. If the view type of the list view control is icon view, small icon view, or report view, this value specifies the number of pixels to scroll. If the view type of the list view control is list view, this value specifies the number of columns to scroll.

dy
Integer value that specifies the amount of vertical scrolling.
If the view type of the list view control is icon view, small icon view, or list view, this value specifies the number of pixels to scroll. If the view type of the list view control is report view, this value specifies the number of lines to scroll.

Return Value (Boolean)

lvmSETBKCOLOR Returns TRUE if successful or FALSE otherwise.
Sets the background color of a list view control.

object.lvmSETBKCOLOR(clrBk)

Parameters

clrBk

Background color to set or the CLR_NONE value for no background color. List view controls with background colors redraw themselves significantly faster than those without background colors.

Return Value (Boolean)

lvmSETCALLBACK_MASK Returns TRUE if successful or FALSE otherwise.
Changes the callback mask for a list view control.

object.lvmSETCALLBACKMASK (mask As Long)

Parameters

mask

Specifies the value of the callback mask. The bits of the mask indicate the item states or images for which the application stores the current state data. This value can be any combination of the following constants:

LVIS_CUT The item is marked for a cut-and-paste operation.

LVIS_DROPHILITED The item is highlighted as a drag-and-drop target.

LVIS_FOCUSED The item has the focus.

LVIS_SELECTED The item is selected.

LVIS_OVERLAYMASK The application stores the image list index of the current overlay image for each item.

LVIS_STATEIMAGEMASK The application stores the image list index of the current state image for each item.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

Remarks

The callback mask of a list view control is a set of bit flags that specify the item states for which the application, rather than the control, stores the current data. The callback mask applies to all of the control's items, unlike the callback item designation, which applies to a specific item. The callback mask is zero by default, meaning that the list view control stores all item state information. After creating a list view control and initializing its items, you can use this method or `lvmSETCALLBACKMASK` message to change the callback mask. To get the current callback mask, send the `lvmGETCALLBACKMASK` message.

See Also

`lvmGETCALLBACKMASK`, `lvmGETDISPINFO`

`lvmSETCOLUMN`

Sets the attributes of a list view column.

`object.lvmSETCOLUMN(Col As Long, mask As Long, fmt As Long, cx As Long, pszText As String, SubItem As Long)`

Parameters

see `lvmGETCOLUMN`

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

`lvmSETCOLUMN_`
`WIDTH`

Changes the width of a column in report or list view.

`object.lvmSETCOLUMNWIDTH (Col As Long, cx As Long)`

Parameters

`Col`

Index of the column. In list view, the `Col` parameter must be -1.

`cx`

New width of the column, in list view coordinates, or one of the following values:

`LVSCW_AUTOSIZE`

Automatically sizes the column.

`LVSCW_AUTOSIZE_USEHEADER`

Automatically sizes the column to fit the header text.

`lvmSETIMAGELIST`

Assigns an image list to a list view control.

`object.lvmSETIMAGELIST (himl As Long, Type As Long)`

Parameters

`himl`

Handle to the image list to assign.

Type

Type of image list. This parameter can be one of the following values:

LVSIL_NORMAL	Image list with large icons
LVSIL_SMALL	Image list with small icons
LVSIL_STATE	Image list with state images

Return Value (Long)

Returns the handle of the image list previously associated with the control if successful; NULL otherwise.

lvmSETITEM

Sets some or all of a list view item's attributes.

object.lvmSETITEM (mask As Long, item As Long, SubItem As Long, state As Long, stateMask As Long, pszText As String, Image As Long, lParam As Long)

Parameters

see lvmGETITEM

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

lvmSETITEMCOUNT

Prepares a list view control for adding a large number of items.

object.lvmSETITEMCOUNT (cItems As Long)

Parameters

cItems

Number of items that the list view control will ultimately contain.

Return Value

No return value.

Remarks

By using this before adding a large number of items, you enable a list view control to reallocate its internal data structures only once rather than every time you add an item.

lvmSETITEMPOSITION

Moves an item to a specified position in a list view control, which must be in icon or small icon view.

object.lvmSETITEMPOSITION (item As Long, x As Long, y As Long)

Parameters

index

Index of the list view item.

x and y

New position of the item's upper-left corner, in view

coordinates.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

Remarks

If the list view control has the LVS_AUTOARRANGE style, the list view control is arranged after the position of the item is set.

lvmSETITEM_ POSITION32

Moves an item to a specified position in a list view control, which must be in icon or small icon view.

object.lvmSETITEMPOSITION (item As Long, x As Long, y As Long)

Parameters

item

Index of the list view item to set the position of.

x and y

New horizontal and vertical coordinates of the item.

Return Value

No return value.

lvmSETITEMSTATE

Changes the state of an item in a list view control.

object.lvmSETITEMSTATE(item As Long, state As Long, mask As Long)

Parameters

item

Index of the list view item.

state

New state bits for the item.

mask

Mask specifying which of the item's current state bits to change.

Return Values

No return value.

See Also

[LVM_SETITEMSTATE](#)

lvmSETITEMTEXT

Changes the text of a list view item or subitem.

object.lvmSETITEMTEXT(item As Long, SubItem As Long, pszText As String)

Parameters

item

Index of the list view item.

iSubItem

Index of the subitem or zero to set the item label.

pszText
String that contains the new text. This parameter can be NULL.

Return Value
No return value.

lvmSETTEXTBKCOLOR Sets the background color of text in a list view control.

object.lvmSETTEXTBKCOLOR(clrText As Long)

Parameters
clrText
New text color.

Return Value (Boolean)
Returns TRUE if successful or FALSE otherwise.

lvmSETTEXTCOLOR Sets the text color of a list view control.

object.lvmSETTEXTCOLOR(clrText As Long)

Parameters
clrText
New text color.

Return Value (Boolean)
Returns TRUE if successful or FALSE otherwise.

lvmSORTITEMS An application-defined comparison function to sort the items of a list view control. The index of each item changes to reflect the new sequence.

object.lvmSORTITEMS (pfnCompare As Long, IParamSort As Long)

Parameters
pfnCompare
Pointer to the application-defined comparison function. The comparison function is called during the sort operation each time the relative order of two list items needs to be compared. Use AddressOf to specify the function address.

IParamSort
Application-defined value that is passed to the comparison function.

Return Value (Boolean)
Returns TRUE if successful or FALSE otherwise.

Remarks
The comparison function has the following form:

Function CompareFunc(IParam1 As Long, IParam2 As Long, IParamSort As Long) As Long

The IParam1 parameter is the 32-bit value associated

with the first item being compared; and the lParam2 parameter is the value associated with the second item. These are the values that were specified in the lParam Parameter when they were inserted into the list. The lParamSort parameter is the same value passed to the lvmSORTITEMS message.

The comparison function must return a negative value if the first item should precede the second, a positive value if the first item should follow the second, or zero if the two items are equivalent.

lvmUPDATE

Updates a list view item. If the list view control has the lvsAUTOARRANGE style, this causes the list view control to be arranged.

object.lvmUPDATE (item As Long)

Parameters

iItem

Index of the item to update.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

Refresh

Standard Method

Move

Standard Method

OLEDrag

SetFocus

Standard Method

Zorder

Standard Method

Events (List View Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Event Description	
Click	Standard Event	
DbClick	Standard Event	
KeyDown	Standard Event	
KeyPress	Standard Event	
KeyUp	Standard Event	
MouseDown	Standard Event	
MouseMove	Standard Event	
MouseUp	Standard Event	
AfterLabelEdit		
BeforeLabelEdit		
DragDrop	Standard Event	
DragOver	Standard Event	
GotFocus	Standard Event	
ItemClick		
LostFocus	Standard Event	

Panel Constituent

Properties

Methods

Events



Class: hbbPanel

Sample-Panel\B3Panel.vbp

Overview

The panel constituent is a control that can be used by itself or used as a container for other controls. The panel has Bevel-in and Bevel-Out properties that are used to change the look of the control. The control has a .Picture property and can be used for displaying a bitmap.

Properties (Panel Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Description	
Alignment	Sets or returns the text alignment within the window. Possible Values: 0 - Left 1 - Right 3 - Center	
AutoSize	When set to True sizes the control to fit the picture.	
BackColor	Standard Property	
BevelIn	Type of inside bevel. Possible Values: 0 - Raised 1 - Lowered	
BevelInWidth	Width of the inside bevel in pixels.	
BevelOut	Type of outside bevel . 0 - Raised 1 - Lowered	
BevelOutWidth	Width of the outside bevel in pixels.	
Border	True / False, turns the border on or off, the border is 1 pixel wide.	
BorderColor	Standard Property	
Caption	Standard Property	
Enabled	Standard Property	
Font	Standard Property	
ForeColor	Standard Property	
hWnd	Standard Property	
LightColor	Used for the light color in 3-d effects.	
MouseIcon	Standard Property	
MousePointer	Standard Property	
Multiline	True / False, turns multiline on or off. Valignment will only work when multiline is False.	
Picture	A picture, if specified is drawn as the background of the control.	
ShadowColor	Color used as the shadow in 3-d effects.	
Valignment	Vertical alignment of the text. Multiline must be false for this to work. 0 - Top 1 - Bottom 2 - Vertical Center	

Methods (Panel Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Description	
AboutBox	Invokes the control's About Box	

Events (Panel Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Description	
Click	Standard Event	
DbClick	Standard Event	
KeyDown	Standard Event	
KeyPress	Standard Event	
KeyUp	Standard Event	
MouseDown	Standard Event	
MouseMove	Standard Event	

Progress Constituent

Properties

Methods

Events



Class: hhhProgressBar

Sample-Progress\B3ProgressBar.vbg

Overview

The progress bar allows you to graphically show the user the progress of an operation. This control allows you to set the minimum and maximum positions for the progress bar, set the current position and also allows for incremental progress (5% at a time, etc).

Properties (Progress Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
Border	True / False, turns the border on or off, the border is 1 pixel wide.	
Enabled	Standard Property	
hWnd	Standard Property	
MouseIcon	Standard Property	
MousePointer	Standard Property	

Methods (Progress Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Method Description	
AboutBox pbmDELTAPOS	Invokes the control's About Box advances the current position of a progress bar by a specified increment and redraws the bar to reflect the new position. object.pbmDELTAPOS (nIncrement As Integer) As Integer Parameters nIncrement - Amount to advance the position. Return Values Returns the previous position.	
pbmSETPOS	sets the current position for a progress bar and redraws the bar to reflect the new position. object.pbmSETPOS (nNewPos As Integer) As Integer Parameters nNewPos - New position. Return Values Returns the previous position.	
pbmSETRANGE	sets the minimum and maximum values for a progress bar and redraws the bar to reflect the new range. object.pbmSETRANGE (nMinRange As Integer, nMaxRange As Integer) As Integer Parameters nMinRange - Minimum range value. By default, the minimum value is zero. nMaxRange - Maximum range value. By default, the maximum value is 100. Return Values Returns the previous range values if successful, or zero otherwise. The low-order word specifies the previous minimum value, and the high-order word specifies the previous maximum value.	
pbmSETSTEP	specifies the step increment for a progress bar. The step increment is the amount by which the progress bar increases its current position whenever it receives a pbmSTEPIT method. By default, the step increment is set to 10. object.pbmSETSTEP (nStepInc As Integer) As Integer lParam = 0;	

pbmSTEPIT

Parameters

nStepInc - New step increment.

Return Values

Returns the previous step increment.

advances the current position for a progress bar by the step increment and redraws the bar to reflect the new position.

An application sets the step increment by using the SETSTEP message.

object.pbmSTEPIT () As Integer

Return Values

Returns the previous position.

Remarks: When the position exceeds the maximum range value, this message resets the current position so that the progress indicator starts over again from the beginning.

Events (Progress Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Event Description	
Click	Standard Event	
DbClick	Standard Event	
MouseDown	Standard Event	
MouseMove	Standard Event	
MouseUp	Standard Event	

ReBar Constituent

Properties

Methods

Events



Class: hhhReBar

Sample-ReBar\bReBarTst.vbg

Overview

The ReBar is a control that can contain a number of bands. Each of these bands can contain one or more controls. The best example of a ReBar is contained within Microsoft's Internet Explorer. At the top of the browser is ReBar control that allows you to enter the location of the web page that would like to view.

If you take a close look at this control, you will notice that it is comprised of a number of bands. The first Band contains the text box in which you enter the web location. There is also a slider that allows you view specific features provided by the web browser. If you slide this slider to the left, you are able to view these choices. If you have Internet Explorer, take a moment to look at this control.

The following section outlines the features of the ReBar constituent.

Features

The ReBar control does not have a standard control counterpart. This section outlines some of the features of the ReBar.

Add and remove bands from the ReBar

The rbINSERTBAND and rbDELETEBAND methods are used to add and remove bands from the ReBar. See the reference section later in this chapter for more information about these methods.

Retrieve the count of bands on the ReBar

The rbGETBANDCOUNT method returns the number of bands that are currently on the ReBar. This method takes no parameters and returns the number of bands that the control contains.

Determine the number of rows the ReBar has

When a user drags a control downward on the ReBar, a new row is created. If the control is dragged back up, the row is destroyed. The rbGETROWCOUNT method returns the number of rows that the ReBar currently contains.

Determine the height of a row on the ReBar

The rbGETROWHEIGHT method returns the height of the specified row (in pixels).

Use ToolTips on the ReBar

ToolTips are set using the rbsTOOLTIPS method. This method

Properties (ReBar Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Description	
rbsTOOLTIPS	Tooltips are supported. This style allows you to use the same tooltip in all of the inner bands. There is no hot region in the ReBar that will otherwise cause the tooltip created by this style bit to be shown.	
RbsVARHEIGHT	The ReBar control displays bands at the minimum required height when possible. Without this style, the ReBar control displays all bands at the same height. The control uses the height of the tallest visible band to determine the height of other bands.	
rbsBANDBORDERS	A border is placed around each band in the ReBar.	
rbsFIXEDORDER	This style prevents the user from reordering the ReBar bands.	

Methods (ReBar Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Description	
rbDELETEDBAND	Removes a band from the ReBar. object.rbDELETEDBAND(Index As Long) As Boolean Parameters Index The zero-based index of the band to delete Returns 0 if successful; non-zero otherwise	
rbGETBANDCOUNT	Returns the number of bands in the ReBar object.rbGETBANDCOUNT() As Long Returns UINT - Number of bands in the ReBar	
rbGETBANDINFO	Retrieves information on the specified band. object.rbGETBANDINFO (Index As Long, fMask As Long, fStyle As Long, clrFore As OLE_COLOR, clrBack As OLE_COLOR, lpText As String, iImage As Long, hwndChild As Long, cxMinChild As Long, cyMinChild As Long, cx As Long, hbmBack As stdPicture, wID As Long) As Boolean Parameters Index - the zero-based index of the place to insert the band; -1 signifies that the band will be added to the end fMask - Flags that indicate which members of this structure are valid or must be filled. This value can be a combination of the following:	
	RBBIM_STYLE	The fStyleMember is valid.
	RBBIM_COLORS	The clrFore and clrBack members are valid
	RBBIM_TEXT	The lpText member is valid.
	RBBIM_IMAGE	The iImage member is valid.
	RBBIM_CHILD	The hwndChild member is valid.

RBBIM_CHILDSize	The cxMinChild and cyMinChild members are valid.
RBBIM_SIZE	The cx member is valid.
RBBIM_BACKGROUND	The hbmBack member is valid.
RBBIM_ID	The wID member is valid.

fStyle - Flags that specify the band style. This value can be a combination of the following:

Value

RBBS_BREAK	The band will be on a new line. Use this style if you plan to have more than one line of controls in your ReBar. This style is used by Internet Explorer 3.0.
RBBS_FIXEDSIZE	The band can't be sized. If you use this style, the sizing grip will not be displayed on the band.
RBBS_CHILDEDGE	An edge will appear around the top and bottom of the child window.
RBBS_HIDDEN	The band will not be visible.
RBBS_NOVERT	The band will not be displayed when the Rebar control is vertical.
RBBS_FIXEDBMP	The bitmap will not move when the band is resized.

clrFore

The band's foreground color, used to draw text.

clrBack

The band's background color.

lpText

The display text for the band.

iImage

Zero-based index of the image that should be displayed in the band, if any. The image list is set using the ImageList property.

hwndChild

Handle of the child window contained in the band, if any.

cxMinChild - The minimum horizontal size of the child

window, in pixels. The band cannot be sized smaller than this value.

cyMinChild - The minimum vertical size of the child window, in pixels. The band cannot be sized smaller than this value.

cx
The horizontal size of the band, in pixels.

hbmBack
The background bitmap.

wID
The ID of the band.

Returns

rbGETROWCOUNT
True / False.
Retrieves the number of rows in the ReBar.

object.rbGETROWCOUNT() As Long

Parameters
None

rbGETROWHEIGHT
Returns
the number of rows
Returns the height of the specified row in the ReBar.

object.rbGETROWHEIGHT(Index As Long) As Long

Parameters
Index
The zero-based row to check

rbINSERTBAND
Returns
the height, in pixels, of the row
Inserts a band into a ReBar.

object.rbGETBANDINFO (Index As Long, fMask As Long, fStyle As Long, clrFore As OLE_COLOR, clrBack As OLE_COLOR, lpText As String, iImage As Long, hwndChild As Long, cxMinChild As Long, cyMinChild As Long, cx As Long, hbmBack As stdPicture, wID As Long) As Boolean

Parameters
See rbGETBANDINFO

Returns

rbSETBANDINFO
True / False.
Sets the information about an existing band in the ReBar.

object.rbGETBANDINFO (Index As Long, fMask As

Long, fStyle As Long, clrFore As OLE_COLOR, clrBack As OLE_COLOR, lpText As String, iImage As Long, hwndChild As Long, cxMinChild As Long, cyMinChild As Long, cx As Long, hbmBack As stdPicture, wID As Long) As Boolean

Parameters

See rbGETBANDINFO

Returns

True / False.

rbSETBARINFO

Sets information about the ReBar. Used right now to activate the image list.

object.rbSETBARINFO () As Boolean

Parameters

None

Returns

True / False.

Events (ReBar Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Description	
HeightChange	<p>HeightChange (newHeight As Single) This event is fired each time the ReBar control resizes for any reason. If you have placed the ReBar control on another control (such as the user control in VB5), be sure to set the height of the containing control in this event.</p> <p>Notes</p> <p>One of the new controls created for Internet Explorer 3.0 is the Rebar control. Rebar contains child windows such as toolbars, combo boxes, and bitmaps, and manages the size and position of the child windows it contains. Once you create the child windows and assign them to the ReBar, the child windows are displayed in it. If you use Internet Explorer 3.0, you see the Rebar control at the top of the screen. You probably thought that is was just a funky toolbar, but it is actually a ReBar. Take a look:</p> <p>Each area that contains a child window is referred to as a band. Rebar controls can contain one or more bands. Each band can have any combination of the following: a gripper bar, a bitmap, a text label, and a child window. However, bands cannot contain more than one of these items. For example, you cannot have two grippers or two child windows within one band.</p> <p>All bands, except those that have the RBBS_FIXEDSIZE style, can be resized via the gripper bar. You can also single-click on the gripper to minimize or maximize the band. The Rebar control manages the size and position of the child window assigned to that band.</p>	

Browser Constituent

Properties

Methods

Events

™

Class: bbbBrowser Sample-Browse\bbbBrowser.vbg

Overview

The Visual Basic 5 environment provides access to an application called the Object Browser. The Object Browser reads type library (typelib) information from an OCX or DLL file. This information includes properties, methods, method parameters, events, constants, etc. The BeCubed Browser is a constituent control that provides access to this functionality from within your Visual Basic program.

The Browser sample program, contained in the *samples\browse* directory of Basic Constituents demonstrates the use of this control.

Open the Browser control group sample. This sample contains 2 projects, the Browser ActiveX sample and a test project. Open the ActiveX sample and then open its user control. Notice that the user control contains, 2 B3ListView controls, a splitter, a text box and a command button.



Figure 1: The Browser ActiveX Control Sample displaying information about the Card constituent.

Note: The B3ListView controls contained within this sample are actually instances of the bbbListView control sample provided with the constituents.

The functionality of the program is quite simple, although the underlying code is a bit more complex. The user clicks the command button (looks like an open file icon). A common dialog box prompts the user for the full path to an .OCX file. Once the user has selected the file, the Browser control adds 4 items to the B3ListView on the left: Properties, Methods, Events and Other. It then enumerates the list of properties for the selected OCX and places them in the B3ListView control on the right. The user may view the methods, events and other information about the control by selecting the corresponding item in the B3ListView on the left.

Under the surface, the Browser constituent invokes a callback procedure that enumerates the properties, methods, events, constants, etc. of the selected control.

Let's take a moment to review how a callback procedure works. A callback is an event procedure that is called repeatedly in order to pass information to the user. In the case of the Browser control, the TypeInfo event is called

repeatedly in order to pass the properties, methods, etc. of the selected control to the user. Each time a the callback occurs, the function is called.

In the case of the Browser constituent, the callback function is a component of the browser (bbbBrowser) control itself and is therefore created when the control is drawn on the form. The Browser's callback is called repeatedly in order to pass the list of properties, methods, etc. of the selected control. In the case of an OCX, the first item that is enumerated is the properties. When the TypeInfo callback is first called, the type parameter indicates what type of item is being passed in. In this case, it is TKIND_PROPERTY, a property. The function is called repeatedly until all of the properties have been enumerated. Once the full list of properties has been passed in, the methods are enumerated. When this begins, the type parameter of the TypeInfo callback indicates that a method is now being passed in. The function continues to be called repeatedly for each of the method's parameters (the type parameter changes to TKIND_PARAMETER to indicate that a parameter is being passed in). This type of iteration continues until all of the properties, methods, events, etc. have been passed to the callback procedure. See the Browser reference later in this chapter for more information about the TypeInfo callback.

Take a moment to look at the TypeInfo procedure of the bbbBrowser control within the ActiveX sample. Set a breakpoint at the beginning of the procedure, close the user control and run the sample project. Click the command button, and from the common dialog box select an OCX file (not COMCTL32.OCX since it contains multiple controls – the sample is not set up to handle this). Once the OCX is selected, the browser constituent begins enumerating the properties, methods events, etc. of the selected control. When the first property is passed to the callback procedure, your breakpoint is hit. Step through the procedure and note the order of events. If the first item is a property (type is TKIND_PROPERTY). It is added to the list of properties in a class called Control, and then the function exits. The function is immediately called again to pass the next property and so on.

It is important to note that the sample contains a class object called objControl. This class is used to store the properties, methods, etc. of the selected control. This is more organized than using arrays. Also, you cannot pass typed arrays to a procedure.

The order of events is as follows:

The user clicks the command button and is prompted to select an OCX. Once selected, the properties, methods, events, etc. of the selected control are enumerated by the browser constituent. As these items are enumerated, they are passed, one by one, to the Browser constituent's callback function. Inside the callback, these items are added to clsControl. This class contains member variables to store the properties, methods, etc. When the user makes a selection in the B3ListView control on the left (chooses which items to view), the corresponding list of items is retrieved from the control class.

This example can be expanded in many ways, including storing the properties, methods, etc. the user has selected from the B3ListView.

Take a look at the Browser test project contained in the project group. This program actually contains one ActiveX control: the Browser ActiveX control. The Browser ActiveX control itself contains the controls necessary to use the browser control. In other words, when you draw the Browser ActiveX control on a form, it already contains all of the controls necessary to utilize the browser's functionality (2 B3ListView's, splitter, text box, command button, etc.).

The browser is an example of a fully functional control that meets a specific need. In order to provide your user with browser capabilities, simply add this ActiveX control to any Visual Basic 5 project.

Features

The Browser contains one major method, GetTypeInfo, This method begins the type library enumeration process based on the selected file set in the TypeLibName property. See the Overview section earlier in this chapter, as well as the reference (following this section) for a complete description of this control.

Properties (Browser Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
Documentation	String that returns the associated documentation string.	
HelpContext		
HelpFile	String that returns the associated help file name.	
Iname	String that returns the associated name.	
IParamType	String that returns the items type.	
TypeLibName	String that sets or returns the Type Lib Name.	

Methods (Browser Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Method Description	
AboutBox	Invokes the control's About Box	
GetTypeInfo	Method that starts the type information process.	
GetDispIProperties		

Events (Browser Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Event Description	
TypeInfo	<p>This event is fired for each item in the type library. From within this event you can use the iName, iParamType, Documentation, and HelpFile string properties to obtain the information about the item.</p> <p>Syntax TypeInfo (item As Long, type As Integer)</p> <p>Parameters item Specifies the index of the item within the object that it exists.</p> <p>type Specifies the type of object that this item represents and will be one of the following values. TKIND_PARAMETER TKIND_PROPERTY</p>	

TKIND_METHOD
TKIND_ENUM
TKIND_RECORD
TKIND_MODULE
TKIND_INTERFACE
TKIND_DISPATCH
TKIND_COCLASS
TKIND_ALIAS
TKIND_UNION

Remarks

No information is available until the TypeLibName has been set with a valid type library file name (or a DLL/OCX that contains a type library).

Once the type library name is specified, but only before the GetTypeInfo method is used, the iName, Documentation and HelpFile strings contain information about the overall type library.

Using the GetTypeInfo method starts a process of enumerating the information in the type library. The TypeInfo event will fire for each item in the type library. The control goes through the type library in a sequential fashion, so associations between items may be based on the type of the item and the order in which it was received. The first TypeInfo event to fire will have a type of TKIND_ENUM, TKIND_RECORD, TKIND_MODULE, TKIND_INTERFACE, TKIND_DISPATCH, TKIND_COCLASS, TKIND_ALIAS or TKIND_UNION. After that you could get a TKIND_METHOD or a TKIND_PROPERTY type of event. This property or event would belong to the last TKIND_ENUM, TKIND_RECORD, etc. that was received.

EnumValue

This event is fired for each enumerated value and provides the value associated with the last item from the TypeInfo event.

Sub object_EnumValue(vValue As Variant)

Parameters:

vValue

Holds the value of the enumerated property. This could be any type that Variant supports.

Shell Constituent

Properties



Class: bbbShell

Methods

Sample-TreeSh\bTreeSh.vbg

Events

Overview

The Shell Name Space constituent is a custom control that allows you to browse objects within the Windows shell.

These objects include:

- The Windows Desktop
- Desktop Folders
- Start Menu Programs
- Startup Folder
- My Computer
- Network Neighborhood
- Control Panel
- Recycle Bin
- Fonts
- Printers

You can use the control in one of three ways:

- Stand-alone
- With a bbbTreeView Control
- With a bbbListView Control
- With both a bbbTreeView and a bbbListView control

The combination of the Shell Name Space, the bbbTreeView and the bbbListView is used to emulate the Windows Explorer. The following section explains the Shell Name Space sample included with Basic Constituents.

Let's take a look at the Shell Name Space sample program. Run Visual Basic and open the Shell Name Space sample. This sample contains the following controls:

- bbbShell
- bbbListView
- bbbTreeView
- bbbSystemImageList
- bbbSplitter
- bbbToolBar
- bbbPanel
- bbbToolTip

These controls are configured to emulate the Windows Explorer. While it is fairly easy to obtain this look using the standard Visual Basic controls, you cannot easily emulate the functionality of Explorer. To solve this problem, the BeCubed controls provide the following extended features:

- Ability to browse items in the Windows Shell (Shell Name Space)
- Ability to access the System Image List (bbbImageList)
- Synchronization of the folder and file listings (bbbTreeView and bbbListView)

Features

This section outlines some of the features that the shell name space constituent provides:

Set the topmost folder for the Shell Name Space

Use the TopFolder method to specify the topmost folder that the user can view. The following line of code sets the topmost folder to the desktop.

```
BBShell11.TopFolder(CSIDL_DESKTOP)
```

See the reference section for more information on this method.

Set the starting folder for the Shell Name Space

The SpecialFolder method is used to set the highest level displayed in the shell name space. This method can be used with any combination of the TreeView, ListView or Shell control by itself. The following line of code sets the starting folder to the Recycle Bin:

```
BBShell11.SpecialFolder CSIDL_BITBUCKET
```

Move up one or more levels in the shell (Must Connect TreeView)

To move up one or more levels, use the `UpLevel` method. This method takes the number of levels to move up as a parameter. The following line of code moves up 2 folders.

```
BBShell111.UpLevel 2
```

Notes

This control can be used by itself. To do that simply use the `SpecialFolder` method to set the starting point. Optionally use the `TopFolder` method to restrict movement past your base folder. The `MoveLevel` and `UpLevel` methods can be used to move from folder to folder.

The folder content information is taken from the `EnumItem` event.

The control can be connected to a Basic Constituent `TreeView` control. In this case the `TreeView` control will be filled with folder information based on the `SpecialFolder` method. The user then can move through the tree control using its functionality. The interaction of the `BeCubed Tree` and the `BeCubed Shell` control will load folders as required.

The control can be connected to a Basic Constituent `ListView` control. In this case the `ListView` control will be filled with folder and object information based on the `SpecialFolder` method. Enabling the `UseListDoubleClick` property will allow the user to navigate to folders shown in the list and the `UpLevel` method can be used to back up levels. The `TopFolder` setting will be honored when using the control this way.

The control can also be connected to both a Basic Constituent `TreeView` and a Basic Constituent `ListView` control. In this case the `TreeView` will be filled with folders based on the `SpecialFolder` method and the `ListView` will be filled based on the focus in the `TreeView` control. The two controls will work together through the shell control to keep the information synched. The user can navigate by using the tree or double clicking in the `ListView` (if the `UseListDoubleClick` property is set).

In all the cases above the `ListView` and `TreeView` can be connected to a Basic Constituent System image list in order to provide the correct icons for the lists in question. Right button context menus are provided by the shell control through both the `TreeView` and the `ListView`. Also Drag source functionality is provided in both the `TreeView` and the list view.

Properties (Shell Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Description	
ListIconSize	Sets the icon size retrieved from the system image list.	
ListSort	Determines the way the items placed in the ListView by the shell control are sorted: 0 - no sorting. 1 - descending. 2 - ascending.	
ListView	Connects a ListView control to the shell control by way of the ListView hWnd.	
Path	Sets or returns a text path (normally used for file system paths). Use this property when ListView and TreeView properties are not set.	
TreeSort	Determines the way the items placed in the TreeView by the shell control are sorted: 0 - no sorting. 1 - descending. 2 - ascending.	
TreeView	Connects a TreeView control to the shell control by way of the TreeView hWnd.	
UseListDoubleClick	If a ListView control is connected to the TreeView this Boolean property determines if double clicking will reload the list with items (in the case of double clicking on a folder) or load the file according to a file association.	

Methods (Shell Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Description	
AboutBox	Displays the control's About Box.	
MoveLevel	MoveLevel (folder As String) this method is used when neither a ListView or TreeView is connected to the shell control to move "down" in the shell structure. folder is specified as a relative folder to the current path.	
SpecialFolder	<p>SpecialFolder (folder As Long) specifies the starting folder within the shell control. You use this method with any combination of TreeView, ListView or the Shell control by itself. The possible values are:</p> <p>CSIDL_BITBUCKET - Recycle bin $\frac{3}{4}$ file system directory containing file objects in the user's recycle bin. The location of this directory is not in the registry; it is marked with the hidden and system attributes to prevent the user from moving or deleting it.</p> <p>CSIDL_COMMON_DESKTOP - File system directory that contains files and folders that appear on the desktop for all users.</p> <p>CSIDL_COMMON_PROGRAMS - File system directory that contains the directories for the common program groups that appear on the Start menu for all users.</p> <p>CSIDL_COMMON_STARTMENU - File system directory that contains the programs and folders that appear on the Start menu for all users.</p> <p>CSIDL_COMMON_STARTUP - File system directory that contains the programs that appear in the Startup folder for all users. The system starts these programs whenever any user logs on to Windows NT or starts up Windows 95.</p> <p>CSIDL_CONTROLS - Control Panel $\frac{3}{4}$ virtual folder containing icons for the control panel applications.</p> <p>CSIDL_DESKTOP - Windows desktop $\frac{3}{4}$ virtual folder at the root of the name space.</p> <p>CSIDL_DESKTOPDIRECTORY - File system directory used to physically store file objects on the desktop (not to be confused with the desktop folder itself).</p> <p>CSIDL_DRIVES - My Computer $\frac{3}{4}$ virtual folder containing everything on the local computer: storage devices, printers, and Control Panel. The folder may also contain mapped network drives.</p> <p>CSIDL_FONTS - Virtual folder containing fonts.</p> <p>CSIDL_NETHOOD - File system directory containing objects that appear in the network neighborhood.</p> <p>CSIDL_NETWORK - Network Neighborhood $\frac{3}{4}$ virtual folder representing the top level of the network hierarchy.</p> <p>CSIDL_PERSONAL - File system directory that serves as a common repository for documents.</p> <p>CSIDL_PRINTERS - Printers folder $\frac{3}{4}$ virtual folder</p>	

containing installed printers.

CSIDL_PROGRAMS - File system directory that contains the user's program groups (which are also file system directories).

CSIDL_RECENT - File system directory that contains the user's most recently used documents.

CSIDL_SENDTO - File system directory that contains Send To menu items.

CSIDL_STARTMENU - File system directory containing Start menu items.

CSIDL_STARTUP - File system directory that corresponds to the user's Startup program group.

CSIDL_TEMPLATES - File system directory that serves as a common repository for document templates.

TopFolder

TopFolder (folder As Long) specifies the level in the shell that the shell control will go 'higher' than this folder. See the SpecialFolder method for the possible values.

UpLevel

UpLevel (levels As Long) this method is used to move "up" the specified number of levels in the shell. It can be used unless a TreeView control is connected to the shell control.

Events (Shell Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name EnumItem	Description EnumItem (DisplayName As String, pidl As String, SmImage As Long, LgImage As Long, SmOpenImage As Long, LgOpenImage As Long) EnumItem is an event used when the control is not connected to a ListView or TreeView. This event is fired each time an item is enumerated and allows the Basic programmer to do whatever he needs to with the information. DisplayName Provides the displayable name of the system object. pidl Pointer to an item identifier list. SmImage Provides the system image index to the small (16 X 16) icon. LgImage Provides the system image index to the large (32 X 32) icon. SmOpenImage Provides the system image index to the small icon that represents the open or selected state. LgOpenImage Provides the system image index to the large icon that represents the open or selected state.	

Splitter Constituent

Properties

Methods

Events



Class: hbbSplit

Sample-Splitter\Splitter.vbg

Overview

A Splitter is a control that is used to separate 2 controls. Using the methods and events of the splitter, the 2 controls can be resized so that they always touch the splitter. The best example of this is the Windows Explorer. The explorer actually contains a TreeView (left side) and a list view (right side). In the center is a splitter control that can be dragged from left to right. As the splitter is dragged, the ListView and TreeView move and resize accordingly.

The splitter has a SizeWindow event that is fired each time the user resizes the splitter. This event provides sufficient information to resize and move the corresponding controls to their new positions. The following reference section provides more information about the properties, methods and events of this control.

Properties (Splitter Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Description	
BorderColor	Standard Property	
FaceColor	Sets the face color (usually light gray).	
LightColor	Sets the light color (usually white).	
MaxPercent	Maximum travel of the splitter, based on a percentage of the client area that the splitter is placed on.	
MinPercent	Minimum travel of the splitter, based on a percentage of the client area that the splitter is placed on.	
ShadowColor	Sets the shadow color (usually dark gray).	
Style	Type of splitter: 0 - Vertical 1 - Horizontal.	

Methods (Splitter Constituent)

Description

Properties

Events

Method Name

Description

UpdateSize

This method is used to force an update of the splitter and avoid an intrusive subclass procedure. Call this method in the splitter container's resize event.

Events (Splitter Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Description	
SizeWindows	<p>SizeWindows (W1Left As Single, W1Top As Single, W1Width As Single, W1Height As Single, W2Left As Single, W2Top As Single, W2Width As Single, W2Height As Single, Cancel As Boolean)</p> <p>The SizeWindows event is fired in order to give the programmer the opportunity to size the windows associated with the splitter.</p>	

Status Bar Constituent

Properties

Methods

Events



Class: hhhStatusBar Sample-Status\Stat.vbg

Overview

A status bar usually appears at the bottom of a form and provides information about the application. The status bar is divided into sections, each containing a piece of information.

The Basic Constituents status bar allows you to add and remove parts (sections of the status bar) and get the dimensions of a part. The following reference covers the properties, methods and events of the status bar.

Properties (Status Bar Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
BackColor	Standard Property	
BorderStyle	Standard Property	
Container	Standard Property	
DragIcon	Standard Property	
DragMode	Standard Property	
Enabled	Standard Property	
ForeColor	Standard Property	
Height	Standard Property	
HelpContextID	Standard Property	
hWnd	Standard Property	
Index	Standard Property	
Left	Standard Property	
MouseIcon	Standard Property	
MousePointer	Standard Property	
Name	Standard Property	
Object	Standard Property	
Parent	Standard Property	
TabIndex	Standard Property	
TabStop	Standard Property	
Tag	Standard Property	
Top	Standard Property	
Visible	Standard Property	
WhatsThisHelpID	Standard Property	
Width	Standard Property	

Methods (Status Bar Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Method Description	
sbGETBORDERS	<p>retrieves the current widths of the horizontal and vertical borders of a status window.</p> <p>object.sbGETBORDERS (nHorizWidth As Long, nVertWidth As Long, nBetweenWidth As Long) As Boolean</p> <p>Parameters nHorizWidth - receives the width of the horizontal border. nVertWidth - receives the width of the vertical border. nBetweenWidth - receives the width of the border between rectangles.</p> <p>Return Values If the operation succeeds, the return value is TRUE. If the operation fails, the return value is FALSE.</p> <p>Remarks The borders determine the spacing between the outside edge of the window and the rectangles within the window that contain text. The borders also determine the spacing between rectangles.</p>	
sbGETPARTS	<p>retrieves a count of the parts in a status window. The message also retrieves the coordinate of the right edge of the specified number of parts.</p> <p>object.sbGETPARTS (nParts As Long, aRightCoord As Long) As Long</p> <p>Parameters nParts - Number of parts for which to retrieve coordinates. If this parameter is greater than the number of parts in the window, the message retrieves coordinates for existing parts only. aRightCoord - this parameter is passed as the first element an integer array that has the same number of elements specified by nParts. Each element in the array receives the client coordinate of the right edge of the corresponding part. If an element is set to - 1, the position of the right edge for that part extends to the right edge of the window. To retrieve the current number of parts, set this parameter to zero.</p> <p>Return Values Returns the number of parts in the window if successful, or zero otherwise.</p>	

sbGETRECT

retrieves the bounding rectangle of a part in a status window.

object.sbGETRECT (iPart As Long, lLeft As Long, lTop As Long, lRight As Long, lBottom As Long) As Boolean

Parameters

iPart - Zero-based index of the part whose bounding rectangle is to be retrieved.

lLeft, lTop, lRight, lBottom - receives the bounding rectangle.

Return Values

If the operation succeeds, the return value is TRUE.

If the operation fails, the return value is FALSE.

sbGETTEXT

message retrieves the text from the specified part of a status window.

object.sbGETTEXT (iPart As Long, iLength As Integer, iType As Integer) As String

Parameters

iPart - Zero-based index of the part from which to retrieve text.

iLength - length, in characters, of the text

iType - the type of operation used to draw the text. The type can be one of the following values:

Value	Meaning
0	The text is drawn with a border to appear lower than the plane of the window.
SBT_NOBORDERS	The text is drawn without borders.
SBT_POPOUT	The text is drawn with a border to appear higher than the plane of window.
SBT_RTLREADING	Displays text using right-to-left reading order on Hebrew or Arabic systems.

If the text has the SBT_OWNERDRAW drawing type, this message returns the 32-bit value associated with the text instead of the length and operation type.

Return Values

The text string.

sbGETTEXTLENGTH

retrieves the length, in characters, of the text from the specified part of a status window.

object.sbGETTEXTLENGTH (iPart As Long) As Long

Parameters

iPart - Zero-based index of the part from which to retrieve text.

Return Values

Returns a 32-bit value that consists of two 16-bit values. The low word specifies the length, in characters, of the text. The high word specifies the type of operation used to draw the text. The type can be one of the following values:

Value	Meaning
0	The text is drawn with a border to appear lower than the plane of the window.
SBT_NOBORDERS	The text is drawn without borders.
SBT_OWNERDRAW	The text is drawn by the parent window.
SBT_POPOUT	The text is drawn with a border to appear higher than the plane of the window.
SBT_RTLEADING	

sbSETMINHEIGHT Displays text using right-to-left reading order on Hebrew or Arabic systems. message sets the minimum height of a status window's drawing area.

object.sbSETMINHEIGHT (minHeight As Short)

Parameters

minHeight - Minimum height, in pixels, of the window.

Return Values

No return value.

Remarks

The minimum height is the sum of minHeight and twice the width, in pixels, of the vertical border of the status window.

sbSETPARTS sets the number of parts in a status window and the coordinate of the right edge of each part.

object.sbSETPARTS (nParts As Long, aWidths As Long) As Boolean

Parameters

nParts - Number of parts to set. The number of parts cannot be greater than 255.

aWidths - first element of an long integer array that has the same number of elements as parts specified by nParts. Each element in the array specifies the position, in client coordinates, of the right edge of the corresponding part. If an element is - 1, the position of the right edge for that part extends to the right edge of the window.

Return Values

If the operation succeeds, the return value is TRUE.

If the operation fails, the return value is FALSE.

sbSETTEXT sets the text in the specified part of a status window.

object.sbSETTEXT (iPart As Long, uType As Long,

szText As String)

Parameters

iPart - Zero-based index of the part to set. If this value is 255, the status window is assumed to be a simple window having only one part.

uType - Type of drawing operation. This parameter can be one of the following values:

Value	Meaning
0	The text is drawn with a border to appear lower than the plane of the window.
SBT_NOBORDERS	The text is drawn without borders.
SBT_OWNERDRAW	The text is drawn by the parent window.
SBT_POPOUT	The text is drawn with a border to appear higher than the plane of the window.
SBT_RTLREADING	Displays text using right-to-left reading order on Hebrew or Arabic systems.

szText - string that specifies the text to set. If uType is SBT_OWNERDRAW, this parameter represents 32 bits of data. The parent window must interpret the data and draw the text when it receives the WM_DRAWITEM message.

Return Values

If the operation succeeds, the return value is TRUE.

If the operation fails, the return value is FALSE.

Remarks

The message invalidates the portion of the window that has changed, causing it to display the new text when the window next receives the WM_PAINT message.

sbSIMPLE

specifies whether a status window displays simple text or displays all window parts set by a previous sbSETPARTS message.

object.sbSIMPLE (fSimple As Boolean) As Boolean

Parameters

fSimple - Display type flag. If this parameter is TRUE, the window displays simple text. If it is FALSE, it displays multiple parts.

Return Values

Returns FALSE if an error occurs.

Remarks

If the status window is being changed from nonsimple to simple, or vice versa, the window is immediately redrawn.

Events (Status Bar Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Event Description	
Click	Standard Event	
DbClick	Standard Event	
MouseDown	Standard Event	
MouseMove	Standard Event	
MouseUp	Standard Event	

System Image List Constituent

Properties

Methods

Events



Class: bbbSysImg

Sample-SysImg\bList.vbg

Overview

The system image list is an image list control that provides access to the set of images that Windows maintains in order to display the proper bitmap for a specific type of object (file, printer, etc.). These images are not normally available to the Visual Basic programmer. Using the BeCubed System Image List Constituent, a Visual Basic program can access these images. The Shell Name Space sample program (provided with the constituents) uses the BeCubed system image list to implement its own version of the Windows explorer.

Properties (System Image Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
BackColor	Standard Property	
hImageList	Handle to the image list. Can be used in conjunction with any Window's API requiring an image list handle.	
IconIndex	Returns the index of the icon associated with the file name last specified in the OnFileName method.	
ImageHeight	Returns the image height.	
ImageWidth	Returns the image width.	
Index	Standard Property	
MaskColor	Standard Property	
Name	Standard Property	
Parent	Standard Property	
Tag	Standard Property	
UseMaskColor		

Methods (System Image Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Method Description	
AboutBox	Invokes the control's About Box.	
Draw	Draws a picture from a system image list to the specified device context. object.Draw(filename As String, hDC As Long, x As Long, y As Long, Style As Long) Parameters filename Specifies the file name. The routine will draw the icon associated with the file name. hDC Specifies the device context to draw the picture. x, y Specifies the top, left coordinate of the picture within the hDC, expressed in pixels, based on window coordinates. style Specifies how the picture is rendered. The following are options. Value Meaning ILD_FOCUS ILD_SELECTED	

ILD_BLEND
ILD_BLEND25 Draws the image, blending 25 percent with the system highlight color. This value has no effect if the image list does not contain a mask.
ILD_BLEND50 Draws the image, blending 50 percent with the system highlight color. This value has no effect if the image list does not contain a mask.
ILD_MASK Draws the mask.
ILD_NORMAL Draws the image using the background color for the image list. If the background color is the CLR_NONE value, the image is drawn transparently using the mask.
ILD_TRANSPARENT Draws the image transparently using the mask, regardless of the background color. This value has no effect if the image list does not contain a mask.
Set the IconIndex property to the icon associated with the specified file name.

OnFileName

object.Draw(filename As String)

Parameters

filename

Specifies the file name. The IconIndex property will be set to the icon that is associated with this file name.

Events (System Image Constituent)

Description

This control has no Events.

Properties

Methods

Tab Constituent

Properties

Methods

Events



Class: bbbTab Sample-TabStrip\Tab.vbg

Overview

A TabStrip is a control that allows you display one or more tabs to the user. Basic Constituent tabs can contain either text or an image. You can also specify that a tab is not to receive the input focus. The following section outlines the features of the TabStrip constituent.

Features

Add a tab to the TabStrip

The tcmINSERTITEM method is used to add tabs to the TabStrip constituent. This method takes the index of the tab, whether the tab displays a text or image heading, the text of the tab (if text is used), an index into an image list (if a picture is used), and an IParam (see the reference for more information).

The following code adds 4 tabs to the TabStrip. The text on each tab is Tab, plus the current index (Tab 1, Tab 2, etc.).

```
Dim lRet As Long
  For iRet = 1 To 4
    lRet = Bbbtab1.tcmINSERTITEM(iRet, TCIF_TEXT, "Tab " & Str$(iRet), 0,
                                0)
  Next iRet
```

Delete all tabs from the TabStrip Constituent

The tcmDELETEALLITEMS method removes all tabs from the TabStrip. This method takes no parameters and returns a Boolean indicating whether or not the removal was successful.

Get the handle of the ImageList control connected to the TabStrip

The tcmGETIMAGELIST method is used to retrieve the handle to the image list. This method takes no parameters and returns the handle to the image list.

Determine the number of rows displayed in the TabStrip Control

The tcmGETROWCOUNT method is used to determine the number of tabs that are currently displayed. This method takes no parameters and returns the number of rows displayed.

The following section outlines the properties, methods and events of the Basic Constituents TabStrip Control.

Properties (Tab Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
BackColor	Standard Property	
BorderStyle	Standard Property	
ccsADJUSTABLE	Enables / disables button dragging on the toolbar.	
Container	Standard Property	
DragIcon	Standard Property	
DragMode	Standard Property	
Enabled	Standard Property	
ForeColor	Standard Property	
Height	Standard Property	
HelpContextID	Standard Property	
hWnd	Standard Property	
Index	Standard Property	
Left	Standard Property	
MouseIcon	Standard Property	
MousePointer	Standard Property	
Name	Standard Property	
Object	Standard Property	
Parent	Standard Property	
TabIndex	Standard Property	
TabStop	Standard Property	
Tag	Standard Property	
tcsBUTTONS	Specifies that tabs appear as buttons and no border is drawn around the display area.	
tcsFIXEDWIDTH	Specifies that all tabs are the same width. This style cannot be combined with the tcsRIGHTJUSTIFY style.	
tcsFOCUSNEVER	Specifies that the tab control never receives the input focus.	
tcsFOCUSONBUTTON_DOWN	Specifies that tabs receive the input focus when clicked.	
tcsFORCEICONLEFT	Aligns icons with the left edge of each fixed-width tab. This style can only be used with the tcsFIXEDWIDTH style.	
tcsFORCELABELLEFT	Aligns labels with the left edge of each fixed-width tab; that is, it displays the label immediately to the right of the icon instead of centering it. This style can only be used with the tcsFIXEDWIDTH style, and it implies the tcsFORCEICONLEFT style.	
tcsMULTILINE	Displays multiple rows of tabs, if necessary, so all tabs are visible at once. tcsOWNERDRAWFIXED - Specifies that the parent window is responsible for drawing tabs. tcsRAGGEDRIGHT - Does not stretch each row of tabs to fill the entire width of the control. This style is the default.	
tcsRIGHTJUSTIFY	Increases the width of each tab, if necessary, so that each row of tabs fills the entire width of the tab control. This window style is ignored unless the TCS_MULTILINE style is also specified. tcsSINGLELINE - Displays only one row of tabs. The user	

can scroll to see more tabs, if necessary. This style is the default.

tcsTABS

Specifies that tabs appear as tabs and that a border is drawn around the display area. This style is the default.

tcsTOOLTIPS - Specifies that the tab control has a tooltip control associated with it. For more information about tooltip controls, see [Tooltip Controls](#)

Top

Standard Property

Visible

Standard Property

WhatsThisHelpID

Standard Property

Width

Standard Property

Methods (Tab Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Method Description	
tcmADJUSTRECT	<p>Calculates a tab control's display area given a window rectangle or calculates the window rectangle that would correspond to a specified display area.</p> <p>object.tcmADJUSTRECT (fLarger As Boolean, iLeft As Long, iTop As Long, iRight As Long, iBottom As Long)</p> <p>Parameters fLarger - Operation to perform. If this parameter is TRUE, prc specifies a display rectangle and receives the corresponding window rectangle. If this parameter is FALSE, prc specifies a window rectangle and receives the corresponding display area. iLeft, iTop, iRight, iBottom - specifies the given rectangle and receives the calculated rectangle.</p> <p>Return Values No return value.</p>	
tcmDELETEALLITEMS	<p>Removes all items from a tab control.</p> <p>object.tcmDELETEALLITEMS () As Boolean</p> <p>Return Values Returns TRUE if successful or FALSE otherwise.</p>	
	tcmDELETEITEM	removes an item from a tab control.
	object.tcmDELETEITEM (iItem As Long) As Boolean	
	<p>Parameters iItem - Index of the item to delete.</p> <p>Return Values Returns TRUE if successful or FALSE otherwise.</p>	
tcmGETCURFOCUS	<p>Returns the index of the item that has the focus in a tab control.</p> <p>object.tcmGETCURFOCUS () As Long</p> <p>Return Values Returns the index of the tab item that has the focus</p> <p>Remarks The item that has the focus may be different than the</p>	

selected item.
tcmGETCURSEL Determines the currently selected tab in a tab control.

object.tcmGETCURSEL () As Long

Return Values
Returns the index of the selected tab if successful or - 1 if no tab is selected.

tcmGETIMAGELIST Retrieves the image list associated with a tab control.

tcmGETIMAGELIST () As Long

Return Values
Returns the handle to the image list if successful or NULL otherwise.

tcmGETITEM Retrieves information about a tab in a tab control.

object.tcmGETITEM (iItem As Long, mask As Long, pszText As String, iImage As Long, lParam As Long) As Boolean

Parameters
iItem - Index of the tab.
mask - Value specifying which members to retrieve or set. This member can be TCIF_ALL (meaning all members), or zero or more of the following values:

Value	Meaning
TCIF_TEXT	The pszText member is valid.
TCIF_IMAGE	The iImage member is valid.
TCIF_PARAM	The lParam member is valid.
TCIF RTLREADING	Displays the text of pszText using right-to-left reading order on Hebrew or Arabic systems.

pszText - Pointer to a null-terminated string that contains the tab text if the structure contains information about a tab. If the structure is receiving information, this member specifies the address of the buffer that receives the tab text.

iImage - Index into the tab control's image list or - 1 if there is no image for the tab.

lParam - Application-defined data associated with the tab. If there are more or less than 4 bytes of application-defined data per tab, an application must define a structure and use it instead of the TC_ITEM structure. The first member of the application-defined structure must be a TC_ITEMHEADER structure.

Return Values
Returns TRUE if successful or FALSE otherwise.

tcmGETITEMCOUNT Retrieves the number of tabs in the tab control.

object.tcmGETITEMCOUNT () As Long

Return Values
Returns the number of items if successful or zero otherwise.

tcmGETITEMRECT
Retrieves the bounding rectangle for a tab in a tab control.

tcmGETITEMRECT (iItem As Long, iLeft As Long, iTop As Long, iRight As Long, iBottom As Long) As Boolean

Parameters
iItem - Index of the tab.
iLeft, iTop, iRight, iBottom - receives the bounding rectangle of the tab, in viewport coordinates.

Return Values
Returns TRUE if successful or FALSE otherwise.

tcmGETROWCOUNT
Retrieves the current number of rows of tabs in a tab control.

object.tcmGETROWCOUNT () As Long

Return Values
Returns the number of rows of tabs.

Remarks
Only tab controls that have the TCS_MULTILINE style can have multiple rows of tabs.

tcmGETTOOLTIPS
Retrieves the handle to the tooltip control associated with a tab control.

object.tcmGETTOOLTIPS () As Long

Return Values
Returns the handle to the tooltip control if successful or NULL otherwise.

Remarks
A tab control creates a tooltip control if it has the TCS_TOOLTIPS style. You can also assign a tooltip control to a tab control by using the TCM_SETTOOLTIPS message.

tcmHITTEST
Determines which tab, if any, is at a specified screen position.

object.tcmHITTEST (x As Long, y As Long, uFlags As Long) As Long

Parameters
x,y - specifies the screen position to test.
uFlags - receives the result of the hit test and can be one of the following values.

Value	Meaning
-------	---------

TCHT_NOWHERE	The position is not over a tab.
TCHT_ONITEM	The position is over a tab, but not over its icon or its text. For owner-drawn tab controls, this value is specified if the position is anywhere over a tab.
TCHT_ONITEMICON	The position is over a tab's icon.
TCHT_ONITEMLABEL	The position is over a tab's text.

Return Values

Returns the index of the tab or - 1 if no tab is at the specified position.

tcmINSERTITEM

Inserts a new tab in a tab control.

object.tcmINSERTITEM (iItem As Long, mask As Long, pszText As String, iImage As Long, lParam As Long) As Long

Parameters

iItem - Index of the new tab.

mask - Value specifying which members to retrieve or set. This member can be TCIF_ALL (meaning all members), or zero or more of the following values:

Value	Meaning
TCIF_TEXT	The pszText member is valid.
TCIF_IMAGE	The iImage member is valid.
TCIF_PARAM	The lParam member is valid.
TCIF_RTLREADING	Displays the text of pszText using right-to-left reading order on Hebrew or Arabic systems.

pszText - Pointer to a null-terminated string that contains the tab text if the structure contains information about a tab. If the structure is receiving information, this member specifies the address of the buffer that receives the tab text.

iImage - Index into the tab control's image list or - 1 if there is no image for the tab.

lParam - Application-defined data associated with the tab. If there are more or less than 4 bytes of application-defined data per tab, an application must define a structure and use it instead of the TC_ITEM structure. The first member of the application-defined structure

must be a TC_ITEMHEADER structure.

Return Values

Returns the index of the new tab if successful or - 1 otherwise.

tcmREMOVEIMAGE

Removes an image from a tab control's image list.

object.tcmREMOVEIMAGE (iImage As Long)

Parameters

iImage - Index of the image to remove.

Return Values

No return value.

Remarks

The tab control updates each tab's image index, so each tab remains associated with the same image it had been. Sets the focus to a specified tab in a tab control.

tcmSETCURFOCUS

object.tcmSETCURFOCUS (iItem As Long)

Parameters

iItem - Specifies the index of the tab that gets the focus.

Return Values

No return value.

Remarks

If the tab control has the TCS_BUTTONS style (button mode), the tab with the focus may be different from the selected tab. For example, when a tab is selected, the user can press the arrow keys to set the focus to a different tab without changing the selected tab. In button mode, TCM_SETCURFOCUS sets the input focus to the button associated with the specified tab, but it does not change the selected tab.

If the tab control does not have the TCS_BUTTONS style, changing the focus also changes selected tab. In this case, the tab control sends the TCN_SELCHANGING and TCN_SELCHANGE notification messages to its parent window.

tcmSETCURSEL

Selects a tab in a tab control.

object.tcmSETCURSEL (iItem As Long) As Long

Parameters

iItem - Index of the tab to select.

Return Values

Returns the index of the previously selected tab if successful or - 1 otherwise.

Remarks

A tab control does not send a TCN_SELCHANGING or TCN_SELCHANGE notification message when a tab is selected using this message.

tcmSETIMAGELIST

Assigns an image list to a tab control.

object.tcmSETIMAGELIST (himl As Long) As Long

Parameters

himl - Handle of the image list to assign to the tab control.

Return Values

Returns the handle to the previous image list or NULL if there is no previous image list.

tcmSETITEM

Sets some or all of a tab's attributes.

object.tcmSETITEM (iItem As Long, mask As Long, pszText As String, iImage As Long, lParam As Long) As Boolean

Parameters

iItem - Index of the tab.

mask - Value specifying which members to retrieve or set. This member can be TCIF_ALL (meaning all members), or zero or more of the following values:

Value	Meaning
TCIF_TEXT	The pszText member is valid.
TCIF_IMAGE	The iImage member is valid.
TCIF_PARAM	The lParam member is valid.
TCIF_RTLREADING	Displays the text of pszText using right-to-left reading order on Hebrew or Arabic systems.

pszText - Pointer to a null-terminated string that contains the tab text if the structure contains information about a tab. If the structure is receiving information, this member specifies the address of the buffer that receives the tab text.

iImage - Index into the tab control's image list or - 1 if there is no image for the tab.

lParam - Application-defined data associated with the tab. If there are more or less than 4 bytes of application-defined data per tab, an application must define a structure and use it instead of the TC_ITEM structure. The first member of the application-defined structure must be a TC_ITEMHEADER structure.

Return Values

Returns TRUE if successful or FALSE otherwise.

tcmSETITEMSIZE

Sets the width and height of tabs in a fixed-width or owner-drawn tab control.

object.tcmSETITEMSIZE (cx As Integer, cy As

Integer) As Long

Parameters

cx, cy - New width and height, in pixels.

Return Values

Returns the old width and height. The width is in the low-order word of the return value, and the height is in the high-order word.

tcmSETPADDING

Sets the amount of space (padding) around each tab's icon and label in a tab control.

object.tcmSETPADDING (cx As Integer, cy As Integer)

Parameters

cx, cy - Amount of horizontal and vertical padding, in pixels.

Return Values

No return value.

tcmSETTOOLTIPS

Assigns a tooltip control to a tab control.

tcmSETTOOLTIPS (hWnd As Long)

Parameters

hwndTT - Handle to the tooltip control.

Return Values

No return value.

Remarks

You can get the tooltip control associated with a tab control by using the TCM_GETTOOLTIPS message.

Events (Tab Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Description	
Click	Standard Event	
DbClick	Standard Event	
KeyDown	Standard Event	
KeyPress	Standard Event	
KeyUp	Standard Event	
MouseDown	Standard Event	
MouseMove	Standard Event	
MouseUp	Standard Event	
SelChange	This event is fired after the selected tab changes.	
	<i>Sub object_SelChange()</i>	
SelChanging	This event is fired just before the selected tab changes.	
	<i>Sub object_SelChanging()</i>	

Tip Constituent

Properties

Methods

Events



Class: hbbTipsSample-Tips\Tips.vbp

Overview

A tool tip is a small window that floats above a form or control. Tool tips are generally used to display helpful text to the user. The Basic Constituents tool tip control is capable of managing tool tips for multiple controls simultaneously.

Features

This section outlines some of the features of the Basic Constituents tool tip.

Activate or deactivate the tooltip

The ttmACTIVATE method is used to activate and deactivate tool tips. This method takes a boolean indicating whether or not to activate the control.

Add a tool tip to the control

The tool tip control can manage multiple tool tips. Each of these tips is identified by the uid parameter of the ttmADDTOOL method. This method is used register a new tool with the tool tip control. A tool is essentially a new tool tip window that is displayed over a form or control. See the reference section for more information about this method.

Remove a tool from the tool tip

The ttmDELTOOL method removes a tool from the tool tip. See the reference section for more information about this method.

Retrieve information about the current tool

The ttmGETCURRENTTOOL method retrieves information about the current tool. The current tool is the one that the tool tip control is currently displaying text for. See the reference section for more information about this method.

Determine the number of tools maintained by the tool tip

The ttmGETTOOLCOUNT method is used to count the number of tools currently maintained by the tooltip. See the reference section for more information about this method.

Set the size of a tool window

The ttmNEWTOOLRECT method is used to set the bounding rectangle of a tool maintained by the tooltip. This method takes the handle and uId, along with the new coordinates for the tool.

Properties (Tip Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Description	
Enabled	Standard Property	
hWnd	Standard Property	
Index	Standard Property	
Name	Standard Property	
Tag	Standard Property	
ttsALWAYSTIP	A tooltip control with this style appears when the cursor is on a tool, regardless of whether the tooltip control's owner window is active or inactive. Without this style, the tooltip control appears when the tool's owner window is active, but not when it is inactive.	
ttsNOPREFIX	prevents the system from stripping the ampersand (&) character from a string. If a tooltip control does not have the this style, the system automatically strips ampersand characters, allowing an application to use the same string as both a menu item and as text in a tooltip control.	
Visible	Standard Property	
WhatsThisHelpID	Standard Property	

Methods (Tip Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Description	
Refresh ttmACTIVATE	Standard Method Activates or deactivates a tooltip control. object.ttmACTIVATE (fActivate As Boolean) Parameters fActivate Activation flag. If this parameter is TRUE, the tooltip control is activated. If it is FALSE, the tooltip control is deactivated. Return Values No return value.	
ttmADDTOOL	Registers a tool with a tooltip control. object.ttmADDTOOL (uFlags As Long, hWnd As Long, uId As Long, iLeft As Long, iTop As Long, iRight As Long, iBottom As Long, lpszText As String) As Boolean Parameters uFlags A set of bit flags. This member can be a combination of the following values:	
	TTF_IDISHWND	Indicates that the uId member is the window handle to the tool. If this flag is not set, uId is the identifier of the tool.
	TTF_CENTERTIP	Centers the tooltip window below the tool specified by the uId member.
	TTF_RTLREADING	Displays text using right-to-left reading order on Hebrew or Arabic systems.
	TTF_SUBCLASS	Indicates that the tooltip control should subclass the tool's window to intercept messages, such as

WM_MOUSEMOVE If not set, you need to use the TTM_RELAYEVENT message to forward messages to the tooltip control. For a list of messages that a tooltip control processes, see TTM_RELAYEVENT.

hWnd

Handle to the window that contains the tool. If lpszText includes the LPSTR_TEXTCALLBACK value, this member identifies the window that receives TTN_NEEDTEXT notification messages.

uId

Application-defined identifier of the tool. If uFlags includes the TTF_IDISHWND value, uId must specify the window handle to the tool.

iLeft, iTop, iRight, iBottom

Coordinates of the bounding rectangle of the tool. The coordinates are relative to the upper-left corner of the client area of the window identified by hWnd. If uFlags includes the TTF_IDISHWND value, this member is ignored.

lpszText

Pointer to the buffer that contains the text for the tool, or identifier of the string resource that contains the text. If this member is set to the LPSTR_TEXTCALLBACK value, the control sends the TTN_NEEDTEXT notification message to the owner window to retrieve the text.

Return Values

Returns TRUE if successful or FALSE otherwise.

ttmDELTOOL

Removes a tool from a tooltip control.

object.ttmDELTOOL (hWnd As Long, uId As Long)

Parameters

hWnd

Handle to the window that contains the tool. If lpszText includes the LPSTR_TEXTCALLBACK value, this member identifies the window that receives TTN_NEEDTEXT notification messages.

uId

Application-defined identifier of the tool. If uFlags includes the TTF_IDISHWND value, uId must specify the window handle to the tool.

Return Values

`ttmGETCURRENTTOOL` No return value.
Retrieves the information that a tooltip control maintains about the current tool; that is, the tool for which the tooltip is currently displaying text.

`object.ttmGETCURRENTTOOL` (`iTool` As Long, `uFlags` As Long, `hWnd` As Long, `uId` As Long, `iLeft` As Long, `iTop` As Long, `iRight` As Long, `iBottom` As Long, `lpszText` As String) As Boolean

Parameters

`iTool`

Zero-based index of the tool for which to retrieve information.

`uFlags`

A set of bit flags. This member can be a combination of the following values:

`TTF_IDISHWND` Indicates that the `uId` member is the window handle to the tool. If this flag is not set, `uId` is the identifier of the tool.

`TTF_CENTERTIP` Centers the tooltip window below the tool specified by the `uId` member.

`TTF_RTLREADING` Displays text using right-to-left reading order on Hebrew or Arabic systems.

`TTF_SUBCLASS` - Indicates that the tooltip control should subclass the tool's window to intercept messages, such as `WM_MOUSEMOVE`. If not set, you need to use the `TTM_RELAYEVENT` message to forward messages to the tooltip control. For a list of messages that a tooltip control processes, see `TTM_RELAYEVENT`.

`hWnd`

Handle to the window that contains the tool. If `lpszText` includes the `LPSTR_TEXTCALLBACK` value, this member identifies the window that receives `TTN_NEEDTEXT` notification messages.

`uId`

Application-defined identifier of the tool. If `uFlags` includes the `TTF_IDISHWND` value, `uId` must specify the window handle to the tool.

iLeft, iTop, iRight, iBottom

Coordinates of the bounding rectangle of the tool. The coordinates are relative to the upper-left corner of the client area of the window identified by hWnd. If uFlags includes the TTF_IDISHWND value, this member is ignored.

lpszText

Pointer to the buffer that contains the text for the tool, or identifier of the string resource that contains the text. If this member is set to the LPSTR_TEXTCALLBACK value, the control sends the TTN_NEEDTEXT notification message to the owner window to retrieve the text.

Return Values

Returns TRUE if any tools are enumerated or FALSE otherwise.

ttmGETTEXT

Retrieves the text that a tooltip control maintains for a tool.

object.ttmGETTEXT (hWnd As Long, uId As Long) As String

Parameters

hWnd

Handle to the window that contains the tool. If lpszText includes the LPSTR_TEXTCALLBACK value, this member identifies the window that receives TTN_NEEDTEXT notification messages.

uId

Application-defined identifier of the tool. If uFlags includes the TTF_IDISHWND value, uId must specify the window handle to the tool.

Return Values

Returns the text associated with the tip tool.

ttmGETTOOLCOUNT

Message retrieves a count of the tools maintained by a tooltip control.

Object.TTM_GETTOOLCOUNT As Long

Return Values

Returns a count of tools.

ttmGETTOOLINFO

Retrieves the information that a tooltip control maintains about a tool.

object.ttmGETTOOLINFO (uFlags As Long, hWnd As Long, uId As Long, iLeft As Long, iTop As Long, iRight As Long, iBottom As Long, lpszText As String) As Boolean

Parameters

uFlags

A set of bit flags. This member can be a combination of the following values:

TTF_IDISHWND	Indicates that the uId member is the window handle to the tool. If this flag is not set, uId is the identifier of the tool.
TTF_CENTERTIP	Centers the tooltip window below the tool specified by the uId member.
TTF_RTLREADING	Displays text using right-to-left reading order on Hebrew or Arabic systems.
TTF_SUBCLASS	Indicates that the tooltip control should subclass the tool's window to intercept messages, such as WM_MOUSEMOVE. If not set, you need to use the TTM_RELAYEVENT message to forward messages to the tooltip control. For a list of messages that a tooltip control processes, see TTM_RELAYEVENT.

hWnd

Handle to the window that contains the tool. If lpszText includes the LPSTR_TEXTCALLBACK value, this member identifies the window that receives TTN_NEEDTEXT notification messages.

uId

Application-defined identifier of the tool. If uFlags includes the TTF_IDISHWND value, uId must specify the window handle to the tool.

iLeft, iTop, iRight, iBottom

Coordinates of the bounding rectangle of the tool. The coordinates are relative to the upper-left corner of the client area of the window identified by hWnd. If uFlags includes the TTF_IDISHWND value, this member is ignored.

lpszText

Pointer to the buffer that contains the text for the tool, or identifier of the string resource that contains the text. If

this member is set to the LPSTR_TEXTCALLBACK value, the control sends the TTN_NEEDTEXT notification message to the owner window to retrieve the text.

Return Values

Returns TRUE if successful or FALSE otherwise.

Remarks - on entry the hWnd and uld parameters define the tool.

ttmHITTEST

Tests a point to determine whether it is within the bounding rectangle of the specified tool and, if the point is within, retrieves information about the tool.

object.ttmHITTEST (hWnd As Long, x As Long, y As Long, uFlags As Long, uld As Long, iLeft As Long, iTop As Long, iRight As Long, iBottom As Long, lpszText As String) As Boolean

Parameters

hWnd

Handle to the tool or window with the specified tool.

x,y

Client coordinates of the point to test.

uFlags

A set of bit flags. This member can be a combination of the following values:

TTF_IDISHWND - Indicates that the uld member is the window handle to the tool. If this flag is not set, uld is the identifier of the tool.

TTF_CENTERTIP - Centers the tooltip window below the tool specified by the uld member.

TTF_RTREADING Displays text using right-to-left reading order on Hebrew or Arabic systems.

TTF_SUBCLASS Indicates that the tooltip control should subclass the tool's window to intercept messages, such as WM_MOUSEMOVE. If not set, you need to use the TTM_RELAYEVENT message to forward messages to the tooltip control. For a list of messages that a tooltip control processes, see TTM_RELAYEVENT.

uld

Application-defined identifier of the tool. If uFlags includes the TTF_IDISHWND value, uld must specify the window handle to the tool.

iLeft, iTop, iRight, iBottom

Coordinates of the bounding rectangle of the tool. The coordinates are relative to the upper-left corner of the client area of the window identified by hWnd. If uFlags

includes the TTF_IDISHWND value, this member is ignored.

lpszText

Pointer to the buffer that contains the text for the tool, or identifier of the string resource that contains the text.

Return Values

Returns TRUE if the tool occupies the specified point or FALSE otherwise.

ttmNEWTOLRECT

Sets a new bounding rectangle for a tool.

object.ttmNEWTOLRECT (hWnd As Long, uId As Long, iLeft As Long, iTop As Long, iRight As Long, iBottom As Long)

Parameters

hWnd

Handle to the window that contains the tool. If lpszText includes the LPSTR_TEXTCALLBACK value, this member identifies the window that receives TTN_NEEDTEXT notification messages.

uId

Application-defined identifier of the tool. If uFlags includes the TTF_IDISHWND value, uId must specify the window handle to the tool.

iLeft, iTop, iRight, iBottom

Coordinates of the bounding rectangle of the tool. The coordinates are relative to the upper-left corner of the client area of the window identified by hWnd.

Return Values

No return value.

ttmSETDELAYTIME

Sets the initial, re-show, and autopopup duration's for a tooltip control.

object.ttmSETDELAYTIME (uFlags As Long, iDelay As Long)

Parameters

uFlags

Duration to set. This parameter can be one of the following values:

TTDT_AUTOMATIC - Automatically calculates the initial, re-show, and autopopup duration's based on the value of iDelay.

TTDT_AUTOPOP - Sets the length of time before the tooltip window is hidden if the cursor remains stationary in the tool's bounding rectangle after the tooltip window has appeared.

TTDT_INITIAL - Sets the length of time that the cursor

must remain stationary within the bounding rectangle of a tool before the tooltip window is displayed.

TTDT_RESHOW - Sets the length of the delay before subsequent tooltip windows are displayed when the cursor is moved from one tool to another.

iDelay
New duration, in milliseconds.

Return Values
No return value.

ttmSETTOOLINFO

Sets the information that a tooltip control maintains for a tool.

object.ttmSETTOOLINFO (uFlags As Long, hWnd As Long, uId As Long, iLeft As Long, iTop As Long, iRight As Long, iBottom As Long, lpszText As String)

Parameters

uFlags

A set of bit flags. This member can be a combination of the following values:

TTF_IDISHWND Indicates that the uId member is the window handle to the tool. If this flag is not set, uId is the identifier of the tool.

TTF_CENTERTIP Centers the tooltip window below the tool specified by the uId member.

TTF_RTLREADING Displays text using right-to-left reading order on Hebrew or Arabic systems.

TTF_SUBCLASS Indicates that the tooltip control should subclass the tool's window to intercept messages, such as WM_MOUSEMOVE. If not set, you need to use the TTM_RELAYEVENT message to forward messages to the tooltip control. For a list of messages that a tooltip control processes, see TTM_RELAYEVENT.

hWnd

Handle to the window that contains the tool. If lpszText includes the LPSTR_TEXTCALLBACK value, this member identifies the window that receives TTN_NEEDTEXT notification messages.

uId

Application-defined identifier of the tool. If uFlags includes the TTF_IDISHWND value, uId must specify the window handle to the tool.

iLeft, iTop, iRight, iBottom

Coordinates of the bounding rectangle of the tool. The coordinates are relative to the upper-left corner of the client area of the window identified by hWnd. If uFlags includes the TTF_IDISHWND value, this member is ignored.

lpszText

Pointer to the buffer that contains the text for the tool, or identifier of the string resource that contains the text. If this member is set to the LPSTR_TEXTCALLBACK value, the control sends the TTN_NEEDTEXT notification message to the owner window to retrieve the text.

Return Values

No return value.

ttmUPDATETIPTEXT

The TTM_UPDATETIPTEXT message sets the tooltip text for a tool.

object.ttmUPDATETIPTEXT (hWnd As Long, uId As Long, lpszText As String)

Parameters

hWnd

Handle to the window that contains the tool. If lpszText includes the LPSTR_TEXTCALLBACK value, this member identifies the window that receives TTN_NEEDTEXT notification messages.

uId

Application-defined identifier of the tool. If uFlags includes the TTF_IDISHWND value, uId must specify the window handle to the tool.

lpszText

Pointer to the buffer that contains the text for the tool, or identifier of the string resource that contains the text. If this member is set to the LPSTR_TEXTCALLBACK value, the control sends the TTN_NEEDTEXT notification message to the owner window to retrieve the text.

Return Values

No return value.

ttmWINDOWFROMPOINT

Allows a subclass procedure to cause a tooltip to display text for a window other than the one beneath the mouse cursor.

object.ttmWINDOWFROMPOINT (x As Long, y As Long) As Long

Parameters

x, y

Defines the point to be checked.

Return Values

The return value is the handle to the window that contains the point, or NULL if no window exists at the specified point.

Remarks

This message is intended to be processed by an application that subclasses a tooltip. It is not intended to be sent by an application. A tooltip sends this message to itself before displaying the text for a window. By changing the coordinates of the point specified by *lppt*, the subclass procedure can cause the tooltip to display text for a window other than the one beneath the mouse cursor.

Events (Tip Constituent)

Description

Properties

Methods

There are no events for this control.

Tool Bar Constituent

Properties

Methods

Events



Class: `bbbToolB`

Sample-ToolBar\`b3ToolBar.vbg`

Overview

A toolbar is a strip of buttons that is usually displayed at the top of an application's form. These buttons are used to execute commands of the application. In most cases, these commands correspond with those offered by the application's menus.

The Basic Constituents toolbar allows you to create toolbar buttons using a number of different styles. In addition to the standard push button, you can create a push button that works like a check box (pressed when selected, depressed when not). You can also create a group of buttons that allows only one button in the group to be pressed at a time. You can also create separators between the buttons of the toolbar. The following section outlines some of the custom features of the Basic Constituents toolbar control.

Features

Add one or more buttons to the toolbar

The `tbADDBUTTONS` method is used to add buttons to the toolbar. This method takes the number of buttons to add, the index of the image to display in the control (if an image is used), a command identifier that is used to fire an event when a specific button is pressed, as well as style and other information. See the reference section for more information about this method.

To manage multiple buttons of the toolbar, you can use a typed array that contains the information about the buttons. This method simplifies the tracking of multiple buttons.

Change the image that is displayed in a toolbar button

The `tbCHANGEBITMAP` method allows you to change the image that is displayed in a toolbar button. These images are stored in an image list, so you pass the index of the image to the `tbCHANGEBITMAP` method. This method takes the id of the button to change the bitmap for along with the index of the new bitmap.

Display the customize toolbar dialog box

The `tbCUSTOMIZE` method displays the customize dialog for the toolbar. This method takes no parameters and has no return value.

Remove a button from the toolbar

The `tbDELETE` method is used to remove a button from the toolbar. This method takes the index of the button to remove and returns a boolean indicating if the deletion was successful.

Retrieve the bounding rectangle of a toolbar button

To retrieve the bounding rectangle, use the `tbGETITEMRECT` method. The method takes the index of the button, as well as variables that receive the coordinate of the button.

Determine the number rows of buttons displayed by the toolbar

The `tbGETROWS` method is used to determine the number of rows that the toolbar displays. This method takes no parameters and returns the number of rows displayed by the toolbar.

Retrieve the handle of the tooltip control associated with the toolbar

The `tbGETTOOLTIPS` method is used to get the handle to the tooltip associated with the toolbar. This method takes no parameters and returns the handle. If there is no handle, the method returns `NULL`.

Set the size of the bitmap displayed in a toolbar button

The `tbSETBITMAPSIZE` method sets the size of the bitmapped images to be added to the toolbar. This method

takes 2 parameters: The width and height, in pixels of the bitmap.

Properties (Tool Bar Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Description	
BackColor	Standard Property	
BorderStyle	Standard Property	
ccsADJUSTABLE	Enables / disables button dragging on the toolbar	
Container		
DragIcon	Standard Property	
DragMode	Standard Property	
Enabled	Standard Property	
ForeColor	Standard Property	
Height	Standard Property	
HelpContextID	Standard Property	
hWnd	Standard Property	
Index	Standard Property	
Left	Standard Property	
MouseIcon	Standard Property	
MousePointer	Standard Property	
Name	Standard Property	
Object	Standard Property	
Parent	Standard Property	
TabIndex	Standard Property	
TabStop	Standard Property	
Tag	Standard Property	
tbstyleALTDRAW	Allows the user to change the position of a toolbar button by dragging it while holding down the ALT key. If this style is not specified, the user must hold down the SHIFT key while dragging a button. Note that the ccsADJUSTABLE style must be specified to enable toolbar buttons to be dragged.	
tbstyleTOOLTIPS	Creates a tooltip control that an application can use to display descriptive text for the buttons in the toolbar.	
tbstyleWRAPABLE	Creates a toolbar that can have multiple lines of buttons. Toolbar buttons can "wrap" to the next line when the toolbar becomes too narrow to include all buttons on the same line. Wrapping occurs on separation and non-group boundaries.	
tbstyleTRANSPARENT	Creates a toolbar that is transparent. The parent's background shows through the toolbar.	
tbstyleFLAT	Creates a toolbar that has no bevel edges around the buttons.	
tbstyleNOTEXT	Creates a toolbar with no text.	
Top	Standard Property	
Visible	Standard Property	
WhatsThisHelpID	Standard Property	
Width	Standard Property	

Methods (Tool Bar Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Description	
Refresh tbADDBITMAP	<p>Standard Method Adds one or more images to the list of button images available for a toolbar.</p> <p>object.tbADDBITMAP (nButtons As Long, nID As Long, pic As StdPicture) As Long</p> <p>Parameters nButtons Number of button images in the bitmap.</p> <p>nID Identifier of a standard set of bitmaps. The following values are valid:</p> <p>0 - no standard bitmaps, use pic parameter.</p> <p>IDB_STD_LARGE_COLOR Adds large, color standard bitmaps.</p> <p>IDB_STD_SMALL_COLOR Adds small, color standard bitmaps.</p> <p>IDB_VIEW_LARGE_COLOR Adds large, color view bitmaps.</p> <p>IDB_VIEW_SMALL_COLOR Adds small, color view bitmaps.</p> <p>pic Picture that defines the toolbar</p> <p>Return Values Returns the index of the first new image if successful or -1 otherwise.</p>	
tbADDBUTTONS	<p>Adds one or more buttons to a toolbar.</p> <p>object.tbADDBUTTONS (uNumButtons As Long, iBitmap As Long, idCommand As Long, fsState As Integer, fsStyle As Integer, dwData As Long, iString As Long) As Boolean</p> <p>Parameters uNumButtons Number of buttons to add.</p>	

The following parameters can be a simple variable (if uNumButtons is 1) or the first element of an array. The array elements should be properly filled and have at least as many elements as the uNumButtons parameter.

iBitmap

Zero-based index of button image.

idCommand

Command identifier associated with the button. This identifier is used in a WM_COMMAND message when the button is chosen. If the fsStyle member is the TBSTYLE_SEP value, this member must be zero.
fsState - Button state flags. This member can be a combination of the following values:

TBSTATE_CHECKED	The button has been pressed.
TBSTATE_ENABLED	The button accepts user input.
TBSTATE_HIDDEN	The button is not visible.
TBSTATE_INDETERMINATE	The button is grayed.
TBSTATE_PRESSED	The button is being pressed.
TBSTATE_WRAP	A line break follows the button.

fsStyle

Button style. This member can be a combination of values listed below.

TBSTYLE_BUTTON	Creates a standard push button.
TBSTYLE_CHECK	Creates a button that toggles between the pressed and not pressed states each time the user clicks it. The button has a different background color when it is pressed.

TBSTYLE_CHECKGROUP Creates a check button that stays pressed until another button in the group is pressed.

TBSTYLE_GROUP Creates a button that stays pressed until another button in the group is pressed.

TBSTYLE_SEP Creates a separator, providing a small gap between button groups. A button that has this style does not receive user input.

dwData
Application-defined value.

iString
Zero-based index of button string.

tbADDSTRING Return Values
Returns TRUE if successful or FALSE otherwise.
Adds a new string to the list of strings available for a toolbar.

object.tbADDSTRING (sString As String) As Long

Parameters
sString
The address of a buffer that contains a string to add to the list.

tbAUTOSIZE Return Values
Returns the index of the new string if successful or - 1 otherwise.
Message causes a toolbar to be resized.

object.tbAUTOSIZE

Parameters
N/A

tbBUTTONCOUNT Return Values
N/A
Retrieves a count of the buttons currently in the toolbar.

object.tbBUTTONCOUNT () As Long

Parameters
N/A

tbCHANGEBITMAP
 Return Values
 Returns the count of the buttons.
 Changes the bitmap for a button in a toolbar.

 object.tbCHANGEBITMAP (idButton As Long,
 iBitmap As Long) As Boolean

 Parameters
 idButton - Command identifier of the button that is to
 receive a new bitmap.

 iBitmap
 Zero-based index of an image in the toolbar's image
 list. The system displays the specified image in the
 button.

tbCHECKBUTTON
 Return Values
 Returns TRUE if successful or FALSE otherwise.
 checks or unchecks a given button in a toolbar.

 object.tbCHECKBUTTON (idButton As Long, fCheck
 As Boolean) As Boolean

 Parameters
 idButton
 Command identifier of the button to check.

 fcheck
 Check flag. If this parameter TRUE, the check is added.
 If it is FALSE, the check is removed.

 Return Values
 Returns TRUE if successful or FALSE otherwise.

 Remarks
 When a button has been checked, it appears to have been
 pressed.

tbCOMMANDTOINDEX
 Retrieves the zero-based index for the button associated
 with the specified command identifier.

 object.tbCOMMANDTOINDEX (idButton As Long)
 As Long

 Parameters
 idButton
 Command identifier associated with the button.

tbCUSTOMIZE
 Return Values
 Returns the zero-based index for the button.
 Displays the Customize Toolbar dialog box.

 object.tbCUSTOMIZE

 Parameters
 None

tbDELETEBUTTON
 Return Values
 No return value.
 Deletes a button from the toolbar.

 object.tbDELETEBUTTON (iButton As Long) As Boolean

 Parameters
 iButton
 Zero-based index of the button to delete.

tbENABLEBUTTON
 Return Values
 Returns TRUE if successful or FALSE otherwise.
 Enables or disables the specified button in a toolbar.

 object.tbENABLEBUTTON (idButton As Long, fEnable As Boolean) As Boolean

 Parameters
 idButton
 Command identifier of the button to enable or disable.

 fEnable
 Enable flag. If this parameter is TRUE, the button is enabled. If it is FALSE, the button is disabled.

 Return Values
 Returns TRUE if successful or FALSE otherwise.

tbGETBITMAP
 Remarks
 When a button has been enabled, it can be pressed and checked.
 Retrieves the index of the bitmap associated with a button in a toolbar.

 object.tbGETBITMAP (idButton As Long) As Long

 Parameters
 idButton
 Command identifier of the button whose bitmap index is to be retrieved.

tbGETBITMAPFLAGS
 Return Values
 If successful, the method returns the index of the bitmap.
 If unsuccessful, it returns zero.
 Retrieves the bitmap flags.

 object.tbGETBITMAPFLAGS As Long

tbGETBUTTON
 Return Values
 Returns the TBBF_LARGE value if the display can handle large bitmaps (that is, if the width of the display has at least 120 pixels per logical inch). Otherwise, the return value is zero.
 Retrieves information about the specified button in a

toolbar.

object.tbGETBUTTON (iButton As Long, iBitmap As Long, idCommand As Long, fsState As Short, fsStyle As Short, dwData As Long, iString As Long) As Boolean

Parameters

iButton

Zero-based index of the button for which to retrieve information.

iBitmap

Zero-based index of button image.

idCommand

Command identifier associated with the button. This identifier is used in a WM_COMMAND message when the button is chosen. If the fsStyle member is the TBSTYLE_SEP value, this member must be zero.

fsState

Button state flags. This member can be a combination of the values listed in Toolbar Button States.

fsStyle

Button style. This member can be a combination of values listed in Toolbar Button Styles.

dwData

Application-defined value.

iString

Zero-based index of button string.

Return Values

Returns TRUE if successful or FALSE otherwise.

tbGETBUTTONTEXT

Retrieves the text of a button in a toolbar.

object.tbGETBUTTONTEXT (idButton As Long) As String

Parameters

idButton

Command identifier of the button whose text is to be retrieved.

Return Values

Returns the string for the button if successful. Otherwise, the returned string is 0 in length.

tbGETITEMRECT

Retrieves the bounding rectangle of a button in a toolbar.

object.tbGETITEMRECT (iButton As Long, iLeft As Long, iTop As Long, iRight As Long, iBottom As Long) As Boolean

Parameters

iButton
Zero-based index of the button for which to retrieve information.

iLeft, iTop, iRight, iBottom
receives the coordinates of the bounding rectangle.

Return Values
Returns TRUE if successful or FALSE otherwise.

Remarks
This message does not retrieve the bounding rectangle for buttons whose state is set to the TBSTATE_HIDDEN value.

tbGETROWS
Retrieves the number of rows of buttons in a toolbar with the TBSTYLE_WRAPABLE style.

object.tbGETROWS As Long

Return Values
Returns the number of rows.

tbGETSTATE
Retrieves information about the state of the specified button in a toolbar, such as whether it is enabled, pressed, or checked.

object.tbGETSTATE (idButton As Long) As Integer

Parameters

idButton

Command identifier of the button for which to retrieve information.

Return Values
Returns the button state information if successful or - 1 otherwise. The button state information can be a combination of the following values:

TBSTATE_CHECKED The button has been pressed.

TBSTATE_ENABLED The button accepts user input.

TBSTATE_HIDDEN The button is not visible.

TBSTATE_INDETERMINATE The button is grayed.

TBSTATE_PRESSED The button is being pressed.

tbGETTOOLTIPS
Retrieves the handle to the tooltip control, if any, TBSTATE_WRAP A line break follows the button.

associated with the toolbar.

object.tbGETTOOLTIPS As Long

Return Values

Returns the handle to the tooltip control or NULL if the toolbar has no associated tooltip.

tbHIDEBUTTON

Hides or shows the specified button in a toolbar.

object.tbHIDEBUTTON (idButton As Long, fShow As Boolean) As Boolean

Parameters

idButton

Command identifier of the button to hide or show.

fShow

Show flag. If this parameter is TRUE, the button is hidden. If it is FALSE, the button is shown.

Return Values

Returns TRUE if successful or FALSE otherwise.

tbINDETERMINATE

Sets or clears the indeterminate state of the specified button in a toolbar.

object.tbINDETERMINATE (idButton As Long, fIndeterminate As Boolean) As Boolean

Parameters

idButton

Command identifier of the button whose indeterminate state is to be set or cleared.

fIndeterminate

Indeterminate flag. If this parameter is TRUE, the indeterminate state is set. If it is FALSE, the state is cleared.

Return Values

Returns TRUE if successful or FALSE otherwise.

tbINSERTBUTTON

Inserts a button into a toolbar.

object.tbINSERTBUTTON (iButton As Long, iBitmap As Long, idCommand As Long, fsState As Integer, fsStyle As Integer, dwData As Long, iString As Long) As Boolean

Parameters

iButton

Zero-based index of a button. The message inserts the new button to the left of this button.

iBitmap

Zero-based index of button image.

idCommand

Command identifier associated with the button. This identifier is used in a WM_COMMAND message when the button is chosen. If the fsStyle member is the TBSTYLE_SEP value, this member must be zero.

fsState

Button state flags. This member can be a combination of the values listed in Toolbar Button States.

fsStyle

Button style. This member can be a combination of values listed in Toolbar Button Styles.

dwData

Application-defined value.

iString

Zero-based index of button string.

Return Values

Returns TRUE if successful or FALSE otherwise.

tbISBUTTONCHECKED Determines whether the specified button in a toolbar is checked.

object.tbISBUTTONCHECKED (idButton As Long) As Boolean

Parameters

idButton

Command identifier of the button.

Return Values

Returns nonzero if the button is checked or zero otherwise.

tbISBUTTONENABLED Determines whether the specified button in a toolbar is enabled.

object.tbISBUTTONENABLED (idButton As Long) As Boolean

Parameters

idButton

Command identifier of the button.

Return Values

Returns nonzero if the button is enabled or zero otherwise.

tbISBUTTONHIDDEN Determines whether the specified button in a toolbar is hidden.

object.tbISBUTTONHIDDEN (idButton As Long) As Boolean

Parameters

idButton

Command identifier of the button.

Return Values

tbISBUTTON_
INDETERMINATE Returns nonzero if the button is hidden or zero otherwise.
Determines whether the specified button in a toolbar is
indeterminate.

object.tbISBUTTONINDETERMINATE (idButton As
Long) As Boolean

Parameters

idButton

Command identifier of the button.

Return Values

Returns nonzero if the button is indeterminate or zero
otherwise.

tbISBUTTONPRESSED determines whether the specified button in a toolbar is
pressed.

object.tbISBUTTONPRESSED (idButton As Long) As
Boolean

Parameters

idButton

Command identifier of the button.

Return Values

Returns nonzero if the button is pressed or zero
otherwise.

tbPRESSBUTTON Presses or releases the specified button in a toolbar.

object.tbPRESSBUTTON (idButton As Long, fPress As
Boolean) As Boolean

Parameters

idButton

Command identifier of the button to press or release.

fPress

Press flag. If this parameter is TRUE, the button is
pressed. If it is FALSE, the button is released.

Return Values

Returns TRUE if successful or FALSE otherwise.

tbSAVERESTORE Saves or restores the state of the toolbar.

object.tbSAVERESTORE (fSave As Boolean, hkr As
Long, pszSubKey As String, pszValueName As String)

Parameters

fSave

Save or restore flag. If this parameter is TRUE, the
information is saved. If it is FALSE, it is restored.

hkr

Handle to the registry key.

pszSubKey

Subkey name.

pszValueName
Value name.

Return Values
No return value.

tbSETBITMAPSIZE Sets the size of the bitmapped images to be added to a toolbar.

object.tbSETBITMAPSIZE (dxBitmap As Integer, dyBitmap As Integer) As Boolean

Parameters
dxBitmap
Width, in pixels, of the bitmapped images.

dyBitmap
Height, in pixels, of the bitmapped images.

Return Values
Returns TRUE if successful or FALSE otherwise.

Remarks
The size can be set only before adding any bitmaps to the toolbar. If an application does not explicitly set the bitmap size, the size defaults to 16 by 15 pixels.

tbSETBUTTONSIZE Sets the size of the buttons to be added to a toolbar.

object.tbSETBUTTONSIZE (dxButton As Integer, dyButton As Integer) As Boolean

Parameters
dxButton
Width, in pixels, of the buttons.

dyButton
Height, in pixels, of the buttons.

Return Values
Returns TRUE if successful or FALSE otherwise.

Remarks
The size can be set only before adding any buttons to the toolbar. If an application does not explicitly set the button size, the size defaults to 24 by 22 pixels.

tbSETCMDID Sets the command identifier of a toolbar button.

object.tbSETCMDID (index As Long, cmdId As Long) As Boolean

Parameters
index
Zero-based index of the button whose command identifier is to be set.

CmdId
Command identifier.

Return Values
Returns TRUE if successful or FALSE otherwise.
Sets the parent window for a toolbar.

tbSETPARENT

object.tbSETPARENT (hwndParent As Long)

Parameters
hwndParent
Handle to the new parent window.

Return Values
No return value.
Sets the number of rows of buttons in a toolbar.

tbSETROWS

object.tbSETROWS (cRows As Integer, fLarger As Boolean, iLeft As Long, iTop As Long, iRight As Long, iBottom As Long)

Parameters
cRows
Number of rows requested. The minimum number of rows is one, and the maximum is equal to the number of buttons in the toolbar.

fLarger
Flag that indicates whether to create more rows than requested when the system cannot create the number of rows specified by cRows. If this parameter is TRUE, the system creates more rows. If it is FALSE, the system creates fewer rows.

iLeft, iTop, iRight, iBottom
Receive the bounding rectangle of the toolbar after the rows are set.

Return Values
No return value.

Remarks
Because the system does not break up button groups when setting the number of rows, the resulting number of rows might differ from the number requested.
Sets the state for the specified button in a toolbar.

tbSETSTATE

object.tbSETSTATE (idButton As Long, fState As Integer) As Boolean

Parameters
idButton
Command identifier of the button.

fState
State flags. This parameter can be a combination of the

following values:

TBSTATE_CHECKED	The button has been pressed.
TBSTATE_ENABLED	The button accepts user input.
TBSTATE_HIDDEN	The button is not visible.
TBSTATE_INDETERMINATE	The button is grayed.
TBSTATE_PRESSED	The button is being pressed.
TBSTATE_WRAP	A line break follows the button.

Return Values

Returns TRUE if successful or FALSE otherwise.
Associates a tooltip control with a toolbar.

tbSETTOOLTIPS

object.tbSETTOOLTIPS (hwndToolTip As Long)

Parameters

hwndToolTip
Handle to the tooltip control.

Return Values

No return value.

Remarks

Any buttons added to a toolbar before using the tbSETTOOLTIPS method are not registered with the tooltip control.

Events (Tool Bar Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Description	
Click	Standard Event	
DbClick	Standard Event	
KeyDown	Standard Event	
KeyPress	Standard Event	
KeyUp	Standard Event	
MouseDown	Standard Event	
MouseMove	Standard Event	
MouseUp	Standard Event	
ButtonClick	ButtonClick (iButton As Long)	

Tray Icon Constituent

Properties

Methods

Events



Class: hbbTraySample-TrayIcon\TrayIcon.vbp

Overview

The Tray Icon constituent is a control that displays a small window in the system tray of the task bar. The task bar is the area of the desktop that contains the Start button in Windows 95. The system tray is the area on the opposite side of the Start button that displays the time and one or more icons.

The Tray Icon constituent allows you to add an icon to the system tray from your Visual Basic program. The icon can receive events for your program to handle (Click, DblClick, etc.). You can also set the icon that is displayed in the system tray.

Features

This section outlines the features of the Tray Icon constituent.

Add the tray icon to the system tray.

The AddIcon method is used to add the tray icon to the system tray. This method takes no parameters and returns a boolean indicating if the addition was successful. The icon displayed is the one set in the .icon property of the control.

Remove the icon from the system tray.

The RemoveIcon method is used to remove the tray icon from the system tray. This method takes no parameters and returns a boolean indicating whether or not the icon was removed.

Note: Always use the RemoveIcon method to remove an icon before exiting your program. If you exit your program without calling RemoveIcon, the icon is stranded in the system tray and is not removed until the system is restarted.

Change the icon that is displayed in the system tray

The ChangeIcon method is used to change the icon that is displayed in the system tray. The new icon displayed is the one set in the .icon property of the control.

Determine when a user has performed a mouse event on the icon.

When the user performs a mouse event on the icon displayed in the system tray (click, double-click, etc.) the corresponding event of the tray icon control is called.

Properties (Tray Icon Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
(Name)	Standard Property	
Enabled	Standard Property	
Icon	The icon to display in the task bar.	
Tip	The “flyover” text to display when the mouse passes over your icon.	

Methods (Tray Icon Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Method Description	
Aboutbox	Invokes the control’s About Box	
AddIcon	Adds the icon to the task bar	
ChangeIcon	Changes the icon currently in the taskbar	
RemoveIcon	Removes the icon from the task bar	

Events (Tray Icon Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Event Description	
Click	Standard Event	
DbClick	Standard Event	
MouseDown	Standard Event	
MouseMove	Standard Event	
MouseUp	Standard Event	

Tree View Constituent

Properties

Methods

Events



Class: bbbTree

Sample-Tree\bbbTreeView.vbg

Overview

A TreeView is a control that displays a list of items in a hierarchical manner. The Windows Explorer contains a standard TreeView control (left side).

The Basic Constituents TreeView control contains a number of features not offered in the standard TreeView. Among them are the ability to get a handle to the edit control of an item and to programmatically instigate the editing of an item. The TreeView allows you to search for a specific item and provides a callback to a user-defined function for custom sorting. You can also trap all of the Windows messages that are sent to the control (see the reference for more information about this topic).

The standard TreeView holds its items in a collection of node objects. These node objects are a part of the standard control, not the base Windows class. Therefore, the Basic constituent implementation does not contain node objects or the nodes collection. The sample program implements these object in case you choose to use them.

For more information, see the ListView chapter earlier in this manual. This chapter explains the constituent's re-implementation of the ListItem object and ListItems collection, and discusses why you may not necessarily want to include them in your ActiveX control.

Features

This section outlines the features of the Basic Constituents TreeView control. For readability, only items that are not available in the standard TreeView control are covered.

Programmatically begin the editing of an item's label

The tvmEDITLABEL method is used to begin the in-place editing of an item's text. When this method is called, the return is the handle to the edit control used to edit the item's text. This method triggers the BeginLabelEdit in the control. See the reference for more information on this method. The tvmENDLABELEDITNOW method ends the editing of the items label.

Retrieve the amount that child items are indented

The tvmGETINDENT method retrieves the number of pixels that child items are indented, relative to their parents. This method takes one parameter (the index of the item) and returns the number of pixels that the item is indented.

Retrieve the incremental search string for the control.

The TreeView constituent maintains an incremental search string of characters entered by a user. The incremental search string is the character sequence the user types while the TreeView has the input focus.

For example, imagine that the following items are in your TreeView:

James
Joan
Joanie

When the user types "J", the list scrolls to the first entry in the TreeView that begins with J (James). At this point, the incremental search string of the TreeView control contains the letter "J". If the user quickly types the letter "o", the incremental search string is updated to contain "Jo" and the list is scrolled to Joan. If the user pauses too long, the incremental search string times out and is reset. Each time the a new character is typed (within the timeout period), the incremental string is updated and the TreeView scrolls to the item that most closely matches the string.

To retrieve the incremental search string, use the `tvmGETISEARCHSTRING` method. This method requires no parameters and returns either the search string, or `NULL` if the control is not in incremental search mode (the `TreeView` does not have the input focus).

Retrieve the bounding rectangle of an item

The `tvmGETITEMRECT` method retrieves the bounding rectangle of an item and indicates whether the item is visible. See the reference for more information about this method.

Determine the number of items that are currently visible

The `tvmGETVISIBLECOUNT` method returns the number of items that are currently visible in the `TreeView`. This method takes no parameters and returns the number of visible items.

Use a custom sorting routine to sort the items in the TreeView

The `tvmSORTCHILDRENCB` method sorts the items of a `TreeView` using an application defined function. This function is passed to the `TreeView` as a parameter to the `tvmSORTCHILDRENCB` method using the `AddressOf` operator.

Properties (Tree View Constituent)

<u>Description</u>	<u>Methods</u>	<u>Events</u>
Property Name	Property Description	
BackColor	Standard Property	
BorderStyle	Standard Property	
Container	Standard Property	
DragIcon	Standard Property	
DragMode	Standard Property	
Enabled	Standard Property	
Font	Standard Property	
ForeColor	Standard Property	
Height	Standard Property	
HelpContextID	Standard Property	
hWnd	Standard Property	
Index	Standard Property	
Left	Standard Property	
Name	Standard Property	
Object	Standard Property	
Parent	Standard Property	
SelectedItem		
TabIndex	Standard Property	
TabStop	Standard Property	
Tag	Standard Property	
Top	Standard Property	
TVS_DISABLEDRA*_ DROP	Prevents the tree-view control from sending TVN_BEGINDRAG notification messages.	
TVS_EDITLABELS	Allows the user to edit the labels of tree-view items.	
TVS_HASBUTTONS	Displays plus (+) and minus (-) buttons next to parent items. The user clicks the buttons to expand or collapse a parent item's list of child items. To include buttons with items at the root of the tree view, TVS_LINESATROOT must also be specified.	
TVS_HASLINES	Uses lines to show the hierarchy of items.	
TVS_LINESATROOT	Uses lines to link items at the root of the tree-view control. This value is ignored if TVS_HASLINES is not also specified.	
TVS_SHOWSEL_ ALWAYS	Causes a selected item to remain selected when the tree-view control loses focus.	
TVS_PRIVATEIMAGE_ LISTS	Causes the control to destroy the image lists associated with the tree when the TreeView is destroyed.	
TVS_NOTOOLTIPS		
TVS_CHECKBOXES		
TVS_TRACKSELECT		
Visible	Standard Property	
WhatsThisHelpID	Standard Property	
Width	Standard Property	

Methods (Tree View Constituent)

<u>Description</u>	<u>Properties</u>	<u>Events</u>
Method Name	Description	
Drag	Standard Method	
Move	Standard Method	
OLEDrag		
Refresh	Standard Method	
SetFocus	Standard Method	
tvmCREATEDRAG_IMAGE	<p>creates a dragging bitmap for the specified item in a tree-view control, creates an image list for the bitmap, and adds the bitmap to the image list. An application can display the image when dragging the item by using the image-list functions.</p> <p>object.tvmCREATEDRAGIMAGE (hitem As Long) As Long</p> <p>Parameters</p> <p>hitem Handle to the item that receives the new dragging bitmap.</p> <p>Return Value (Long) Returns the handle of the image list to which the dragging bitmap was added if successful or NULL otherwise.</p> <p>Remarks</p> <p>If you create a tree-view control without an associated image list, you cannot use the tvmCREATEDRAGIMAGE message to create the image to display during a drag operation. You must implement your own way to support drag and drop cursor.</p>	
tvmDELETEITEM	<p>Removes an item from a tree-view control.</p> <p>object.tvmDELETEITEM (hitem As Long) As Boolean</p> <p>Parameters</p> <p>hitem Handle to the item to delete. If hitem is the TVI_ROOT value, all items are deleted from the tree-view control.</p> <p>Return Value (Boolean) Returns TRUE if successful or FALSE otherwise.</p> <p>Remarks</p> <p>If the item label is being edited, the edit operation is canceled and the parent window receives the TVN_ENDLBELEDIT notification message. The parent window receives a TVN_DELETEITEM notification message when the item is removed.</p>	
tvmEDITLABEL	<p>Begins in-place editing of the specified item's text,</p>	

replacing the text of the item with a single-line edit control containing the text. This message implicitly selects and focuses the specified item.

object.tvmEDITLABEL(hitem As Long) As Long

Parameters

hitem

Handle to the item to edit.

Return Value (Long)

Returns the handle to the edit control used to edit the item text if successful or NULL otherwise.

Remarks

This message triggers a BeginLabelEdit event in the control.

When the user completes or cancels editing, the edit control is destroyed and the handle is no longer valid. You can safely subclass the edit control, but do not destroy it.

The control must have the focus before you send this message to the control. Focus can be set using the SetFocus function.

See Also

[BeginLabelEdit Event](#)

Ends the editing of a tree-view item's label.

tvmENDEDITLABEL_
NOW

object.tvmENDEDITLABELNOW(fCancel As Boolean)
As Boolean

Parameters

fCancel

Variable that indicates whether the editing is canceled without being saved to the label. If this parameter is TRUE, the system cancels editing without saving the changes. Otherwise, the system saves the changes to the label.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

Remarks

This message causes the EndLabelEdit event to fire.

tvmENSUREVISIBLE

Ensures that a tree-view item is visible, expanding the parent item or scrolling the tree-view control, if necessary.

object.tvmENSUREVISIBLE(hitem As Long) As
Boolean

Parameters

hitem

Handle to the item.

Return Value (Boolean)

Returns TRUE if the system scrolled the items in the tree-

view control to ensure that the specified item is visible. Otherwise, the message returns FALSE.

Remarks

If this method expands an item, the control receives the Expand event.

tvmEXPAND

Expands or collapses the list of child items, if any, associated with the specified parent item.

object.tvmEXPAND(hitem As Long, flag As Long) As Boolean

Parameters

flag

Action flag. This parameter can be one of the following values:

- | | |
|-------------------|--|
| TVE_COLLAPSE | Collapses the list. |
| TVE_COLLAPSERESET | Collapses the list and removes the child items. Note that TVE_COLLAPSE must also be specified. |
| TVE_EXPAND | Expands the list. |
| TVE_TOGGLE | Collapses the list if it is currently expanded or expands it if it is currently collapsed. |

Hitem

Handle to the parent item to expand or collapse.

Return Value (Boolean)

Returns TRUE if any change took place or FALSE otherwise.

Remarks

This method does not trigger the Expand or Collapse Events.

tvmGETCOUNT

Retrieves a count of the items in a tree-view control.

object.tvmGETCOUNT() As Long

Return Value (Long)

Returns the count of items.

tvmGETEDITCONTROL

Retrieves the handle to the edit control being used to edit a tree-view item's text.

object.tvmGETEDITCONTROL() As Long

Return Value (Long)

Returns the handle to the edit control if successful or NULL otherwise.

tvmGETIMAGELIST

Retrieves the handle to the normal or state image list associated with a tree-view control.

object.tvmGETIMAGELIST(iImage As Long) As Long

Parameters

iImage

Type of image list to retrieve. This parameter can be one of the following values:

TVSIL_NORMAL Retrieves the normal image list, which contains the selected and unselected images for the tree-view item.

TVSIL_STATE Retrieves the state image list, which contains the images for tree-view items that are in a user-defined state.

Return Values

Returns the handle to the image list.

tvmGETINDENT

Retrieves the amount, in pixels, that child items are indented relative to their parent items.

object.tvmGETINDENT(hitem As Long) As Long

Return Value (Long)

Returns the amount of indentation.

tvmGETISEARCH_STRING

Retrieves the incremental search string for a tree-view control. The tree-view control uses the incremental search string to select an item based on characters typed by the user.

object.tvmGETISEARCHSTRING() As String

Parameters

N/A

Return Value (BSTR)

Returns the incremental search string. If the tree-view control is not in incremental search mode, the return value is NULL.

tvmGETITEM

Retrieves some or all of a tree-view item's attributes.

object.tvmGETITEM(mask As Long, hItem As Long, state As Long, stateMask As Long, pszText As String, cchTextMax As Long, iImage As Long, iSelectedImage As Long, cChildren As Long, lParam As Long) As Boolean

Parameters

mask

Combination of flags that indicate which of the other structure members contain valid data. When this structure is used with the TVM_GETITEM message, the mask member indicates the item attributes to retrieve. This

member can be a combination of the following values:

TVIF_CHILDREN	The cChildren member is valid.
TVIF_HANDLE	The hItem member is valid.
TVIF_IMAGE	The iImage member is valid.
TVIF_PARAM	The lParam member is valid.
TVIF_SELECTEDIMAGE	The iSelectedImage member is valid.
TVIF_STATE	The state and stateMask members are valid.
TVIF_TEXT	The pszText and cchTextMax members are valid.

Hitem

Identifies the item to which this structure refers.

State

Specifies the current state of the item if the item's state is being retrieved, or the new state if the item's state is being set. The stateMask member specifies the bits of the state member that are valid. This member can be any valid combination of state values. For a list of item states, see Tree-View Item States.

StateMask

Specifies the bits of the state member that are valid.

PszText

Pointer to a null-terminated string that contains the item text if the structure specifies item attributes. If this member is the LPSTR_TEXTCALLBACK value, the parent window is responsible for storing the name. In this case, the tree-view control sends the parent window a TVN_GETDISPINFO notification message when it needs the item text for displaying, sorting, or editing, and a TVN_SETDISPINFO notification message when the item text changes.

If the structure is receiving item attributes, this member is the pointer to the buffer that receives the item text.

cchTextMax

Size of the buffer pointed to by the pszText member if the structure is receiving item attributes. If the structure specifies item attributes, this member is ignored.

iImage

Index in the tree-view control's image list of the icon

image to use when the item is in the non-selected state. If this member is the `I_IMAGECALLBACK` value, the parent window is responsible for storing the index. In this case, the tree-view control sends the parent a `TVN_GETDISPINFO` notification message to get the index when it needs to display the image.

`iSelectedImage`

Index in the tree-view control's image list of the icon image to use when the item is in the selected state. If this member is the `I_IMAGECALLBACK` value, the parent window is responsible for storing the index. In this case, the tree-view control sends the parent a `TVN_GETDISPINFO` notification message to get the index when it needs to display the image.

`cChildren`

Flag that indicates whether the item has associated child items. This member is one of the following values:

Value Meaning

zero The item has no child items.

1 The item has one or more child items.

`I_CHILDRENCALLBACK` The parent window keeps track of whether the item has child items. In this case, when the tree-view control needs to display the item, the control sends the parent a `TVN_GETDISPINFO` notification message to determine whether the item has child items.

If the tree-view control has the `TVS_HASBUTTONS` style, it uses this member to determine whether to display the button indicating the presence of child items. You can use this member to force the control to display the button even though the item does not have any child items inserted. This allows you to display the button while minimizing the control's memory usage by inserting child items only when the item is visible or expanded.

`lParam`

A 32-bit value to associate with the item.

Return Value (Boolean)

Returns `TRUE` if successful or `FALSE` otherwise.

Comments

The `TV_ITEM` structure specifies the information to retrieve and receives information about the item. When the message is sent, the `hItem` parameter identifies the item to retrieve information about and the `mask` parameter specifies the attributes to retrieve.

If `mask` specifies the `TVIF_TEXT` flag, the `pszText` parameter must contain the string that receives the item text and the `cchTextMax` member must specify the size of the buffer.

If `mask` specifies the `TVIF_STATE` value, the `stateMask` member indicates which item states are to be returned.

tvmGETITEMstruct

Retrieves some or all of a tree-view item's attributes using a structure rather than parameters.

object.tvmGETITEMstruct(itemmask As Long) As Boolean

Parameters

itemmask

A pointer to the first element in the TV_ITEM structure. This method expects the TV_ITEM structure to exist and by passing the first element of the structure, the method receives a pointer to the structure. The TV_ITEM structure specifies the information to retrieve and receives information about the item. When the message is sent, the hItem member identifies the item to retrieve information about and the mask member specifies the attributes to retrieve.

If mask specifies the TVIF_TEXT value, the pszText member must contain the pointer to the buffer that receives the item text and the cchTextMax member must specify the size of the buffer.

If mask specifies the TVIF_STATE value, the stateMask member indicates which item states are to be returned.

Type TV_ITEM

mask	As Long
hItem	As Long
state	As Long
stateMask	As Long
pszText	As String
cchTextMax	As Long
iImage	As Long
iSelectedImage	As Long
cChildren	As Long
lParam	As Long

End Type

Members

mask

Combination of flags that indicate which of the other structure members contain valid data. When this structure is used with the TVM_GETITEM message, the mask member indicates the item attributes to retrieve. This member can be a combination of the following values:

TVIF_CHILDREN	The cChildren member is valid.
TVIF_HANDLE	The hItem member is valid.
TVIF_IMAGE	The iImage member is valid.
TVIF_PARAM	The lParam member is valid.
TVIF_SELECTEDIMAGE	The iSelectedImage

	member is valid.
TVIF_STATE	The state and stateMask members are valid.
TVIF_TEXT	The pszText and cchTextMax members are valid.

hItem

Identifies the item to which this structure refers.

state

Specifies the current state of the item if the item's state is being retrieved, or the new state if the item's state is being set. The stateMask member specifies the bits of the state member that are valid. This member can be any valid combination of state values. For a list of item states, see Tree-View Item States.

stateMask

Specifies the bits of the state member that are valid.

pszText

Pointer to a null-terminated string that contains the item text if the structure specifies item attributes. If this member is the LPSTR_TEXTCALLBACK value, the parent window is responsible for storing the name. In this case, the tree-view control sends the parent window a TVN_GETDISPINFO notification message when it needs the item text for displaying, sorting, or editing, and a TVN_SETDISPINFO notification message when the item text changes.

If the structure is receiving item attributes, this member is the pointer to the buffer that receives the item text.

cchTextMax

Size of the buffer pointed to by the pszText member if the structure is receiving item attributes. If the structure specifies item attributes, this member is ignored.

iImage

Index in the tree-view control's image list of the icon image to use when the item is in the non-selected state. If this member is the I_IMAGECALLBACK value, the parent window is responsible for storing the index. In this case, the tree-view control sends the parent a TVN_GETDISPINFO notification message to get the index when it needs to display the image.

iSelectedImage

Index in the tree-view control's image list of the icon image to use when the item is in the selected state. If this member is the I_IMAGECALLBACK value, the parent window is responsible for storing the index. In this case, the tree-view control sends the parent a

TVN_GETDISPINFO notification message to get the index when it needs to display the image.

cChildren

Flag that indicates whether the item has associated child items. This member is one of the following values:

zero	The item has no child items.
1	The item has one or more child items.
I_CHILDRENCALLBACK	The parent window keeps track of whether the item has child items. In this case, when the tree-view control needs to display the item, the control sends the parent a TVN_GETDISPINFO notification message to determine whether the item has child items.

If the tree-view control has the TVS_HASBUTTONS style, it uses this member to determine whether to display the button indicating the presence of child items. You can use this member to force the control to display the button even though the item does not have any child items inserted. This allows you to display the button while minimizing the control's memory usage by inserting child items only when the item is visible or expanded.

IParam

A 32-bit value to associate with the item.

Remarks

This structure is used with the TVM_GETITEM, TVM_SETITEM, and TVM_INSERTITEM messages. It is also included with many of the notification messages. When the structure is used to retrieve item information, only the structure members indicated by mask contain valid data. All other members are invalid.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

tvmGETITEMRECT

Retrieves the bounding rectangle for a tree-view item and indicates whether the item is visible.

object.tvmGETITEMRECT(hitem As Long, fItemRect As Boolean, top As Long, left As Long, bottom As Long, right As Long) As Boolean

Parameters

hitem
Handle to an item.

fItemRect
Value specifying the portion of the item for which to retrieve the bounding rectangle. If this parameter is TRUE, the bounding rectangle includes only the text of the item. Otherwise, it includes the entire line that the item occupies in the tree-view control.
top, left, bottom, right
Receives the bounding rectangle. The coordinates are relative to the upper-left corner of the tree-view control.

Return Value (Boolean)
If the item is visible and retrieves the bounding rectangle, the return value is TRUE. Otherwise, the message returns FALSE and does not retrieve the bounding rectangle.
Retrieves the tree-view item that bears the specified relationship to a specified item.

tvmGETNEXTITEM

object.tvmGETNEXTITEM(hitem As Long, flag As Long) As Long

Parameters

flag

Flag specifying the item to retrieve. This parameter can be one of the following values:

TVGN_CARET	Retrieves the currently selected item.
TVGN_CHILD	Retrieves the first child item of the item specified by the hitem parameter.
TVGN_DROPHILITE	Retrieves the item that is the target of a drag-and-drop operation.
TVGN_FIRSTVISIBLE	Retrieves the first visible item.
TVGN_NEXT	Retrieves the next sibling item.
TVGN_NEXTVISIBLE	Retrieves the next visible item that follows the specified item. The specified item must be visible. Use the <u>TVM_GETITEMRECT</u> message to determine whether an item is visible.
TVGN_PARENT	Retrieves the parent of the specified item.
TVGN_PREVIOUS	Retrieves the previous

sibling item.

TVGN_PREVIOUSVISIBLE Retrieves the first visible item that precedes the specified item. The specified item must be visible. Use the [TVM_GETITEMRECT](#) message to determine whether an item is visible.

TVGN_ROOT Retrieves the topmost or very first item of the tree-view control.

hitem
Handle to an item.

Return Value (Long)
Returns the handle to the item if successful or NULL otherwise.

tvmGETVISIBLE_COUNT Obtains the number of items that are fully visible in the client window of a tree-view control.

`object.tvmGETVISIBLECOUNT()` As Long

Return Value (Long)
Returns the number of items that are fully visible in the client window of the tree-view control.

Remarks
Note that the return value is the number of fully-visible items. If you can see all of 20 items, and part of one more item, the return value is 20, not 21.

tvmHITTEST Determines the location of the specified point relative to the client area of a tree-view control.

`object.tvmHITTEST(x As Long, y As Long, flags As Long)` As Long

Parameters
When the method is invoked, the x and y member specifies the Coordinates of the point to test. When the message returns, the return value is the handle to the item at the specified point or NULL if no item occupies the point. Also, when the method returns, the flags parameter is a hit-test value that indicates the location of the specified point.

x, y
Client coordinates of point to test.

flags
Variable that receives information about the results of a hit test. This member can be one or more of the following

values:

TVHT_ABOVE	Above the client area
TVHT_BELOW	Below the client area
TVHT_NOWHERE	In the client area, but below the last item.
TVHT_ONITEM	On the bitmap or label associated with an item
TVHT_ONITEMBUTTON	On the button associated with an item
TVHT_ONITEMICON	On the bitmap associated with an item
TVHT_ONITEMINDENT	In the indentation associated with an item
TVHT_ONITEMLABEL	On the label (string) associated with an item
TVHT_ONITEMRIGHT	In the area to the right of an item
TVHT_ONITEMSTATEICON	On the state icon for a tree-view item that is in a user-defined state
TVHT_TOLEFT	To the right of the client area
TVHT_TORIGHT	To the left of the client area

Return Value (Long)

Returns the handle to the tree-view item that occupies the specified point or NULL if no item occupies the point.

tvmINSERTITEM

Inserts a new item in a tree-view control.

object.tvmINSERTITEM(hParent As Long, hInsertAfter As Long, mask As Long, hItem As Long, state As Long, stateMask As Long, pszText As String, cchTextMax As Long, iImage As Long, iSelectedImage As Long, cChildren As Long, lParam As Long) As Long

Parameters

hParent

Handle to the parent item. If this member is the TVI_ROOT value or NULL, the item is inserted at the root of the tree-view control.

hInsertAfter

Handle to the item after which the new item is to be inserted or one of the following values:

TVI_FIRST	Inserts the item at the beginning of the list.
-----------	--

TVI_LAST Inserts the item at the end of the list.

TVI_SORT Inserts the item into the list in alphabetical order.

Return Value (Long)
Returns the handle to the new item if successful or NULL otherwise.

tvmINSERTITEMstruct inserts a new item in a tree-view control.

object.tvmINSERTITEMstruct(hParentInsert As Long)
As Long

Parameters

hParentInsert

A pointer to the first item in the TV_INSERTSTRUCT structure. This method expects the TV_INSERTSTRUCT structure to exist at that pointer location. The TV_INSERTSTRUCT structure contains information used to add a new item to a tree-view control.

Type TV_INSERTSTRUCT
 hParent As Long
 hInsertAfter As Long
 item As TV_ITEM

End Type

Members

hParent

Handle to the parent item. If this member is the TVI_ROOT value or NULL, the item is inserted at the root of the tree-view control.

hInsertAfter

Handle to the item after which the new item is to be inserted or one of the following values:

TVI_FIRST Inserts the item at the beginning of the list.

TVI_LAST Inserts the item at the end of the list.

TVI_SORT Inserts the item into the list in alphabetical order.

item

Information about the item to add. See tvmGETITEMstruct

Return Value (Long)
Returns the handle to the new item if successful or NULL otherwise.

tvmSELECTITEM Selects the specified tree-view item, scrolls the item into view, or redraws the item in the style used to indicate the

target of a drag-and-drop operation.

object.tvmSELECTITEM(flag As Long, hitem As Long)
As Boolean

Parameters

flag

Action flag. This parameter can be one of the following values:

TVGN_CARET	Sets the selection to the given item.
TVGN_DROPHILITE	Redraws the given item in the style used to indicate the target of a drag and drop operation.
TVGN_FIRSTVISIBLE	Scrolls the tree view vertically so that the given item is the first visible item.

hitem

Handle to an item. If hitem is NULL, the selection is removed from the currently selected item, if any.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

tvmSETIMAGELIST

Sets the normal or state image list for a tree-view control and redraws the control using the new images.

object.tvmSETIMAGELIST(iImage As Long, himl As Long) As Long

Parameters

iImage

Type of image list to set. For a list of possible values, see the description of the tvmGETIMAGELIST message.

himl

Handle to the image list. If himl is NULL, all images are removed from the tree-view control.

Return Value (Long)

Returns the handle to the previous image list, if any, or NULL otherwise.

See Also

[tvmGETIMAGELIST](#)

tvmSETINDENT

Sets the width of indentation for a tree-view control and redraws the control to reflect the new width.

object.tvmSETINDENT(indent As Long) As Long

Parameters

indent

Width, in pixels, of the indentation. If this parameter is less than the system-defined minimum width, the new width is set to the system-defined minimum.

Return Values

No return value.

tvmSETITEM

Sets some or all of a tree-view item's attributes.

object.tvmSETITEM(mask As Long, hItem As Long, state As Long, stateMask As Long, pszText As String, cchTextMax As Long, iImage As Long, iSelectedImage As Long, cChildren As Long, lParam As Long) As Boolean

Parameters

See tvmGETITEM for explanation of parameters.

Return Value (Boolean)

Returns zero if successful or -1 otherwise.

See Also

tvmGETITEM

tvmSETITEMstruct

Sets some or all of a tree-view item's attributes.

object.tvmSETITEMstruct(itemmask As Long) As Boolean

Parameters

See tvmGETITEMstruct for explanation of parameters.

Return Value (Boolean)

Returns zero if successful or -1 otherwise.

See Also

tvmGETITEMstruct

tvmSORTCHILDREN

Sorts the child items of the specified parent item in a tree-view control.

object.tvmSORTCHILDREN(hitem As Long, fRecurse As Boolean) As Boolean

Parameters

fRecurse

Reserved for future use. Must be zero.

hitem

Handle to the parent item whose child items are to be sorted.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

tvmSORTCHILDRENCB

Sorts tree view items using an application defined callback function that compares the items.

object.tvmSORTCHILDRENCB(lpfnCompare As Long, hParent As Long, lParam As Long) As Boolean

Parameters

lpfnCompare

Pointer to an application-defined callback function, which is called during a sort operation each time the relative order of two list items needs to be compared. The callback function has the following form:

```
Function CompareFunc(lParam1 As Long, lParam2 As Long, lParamSort As Long) As Long
```

The callback function must return a negative value if the first item should precede the second, a positive value if the first item should follow the second, or zero if the two items are equivalent.

The lParam1 and lParam2 parameters correspond to the lParam member of the TV_ITEM structure for the two items being compared. The lParamSort parameter corresponds to the lParam parameter passed with this method.

hParent

Handle to the parent item.

lParam

Application-defined 32-bit value.

Return Value (Boolean)

Returns TRUE if successful or FALSE otherwise.

Standard Method

ZOrder

Events (Tree View Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Description	
AfterLabelEdit		
BeforeLabelEdit		
Click	Standard Event	
Collapse		
DblClick	Standard Event	
DragDrop	Standard Event	
DragOver	Standard Event	
Expand		
GotFocus	Standard Event	
ItemClick		
KeyDown	Standard Event	
KeyUp	Standard Event	
KeyPress	Standard Event	
LostFocus	Standard Event	
MouseDown	Standard Event	
MouseUp	Standard Event	
MouseMove		
OLECompleteDrag	Standard Event	
OLEDragDrop	Standard Event	
OLEDragOver	Standard Event	
OLEGiveFeedBack		
OLESetData		
OLEStartDrag		

Zip Constituent

Properties

Methods

Events



Class: bbbZip Sample-Zip\B3Zip.vbp

Overview

The Zip Constituent is a control that is used to create and manage archives of compressed files. Files can be added to the archives and extracted at a later date as needed. This control is a limited functionality edition of the **Compression Plus** control available from **EITech Development**.

An archive is essentially a file that contains other files. When an archive is created, its filename is set using the .ZipFile property. The file to be added to the archive is set using the .FileSpec property. This property allows you to specify multiple file specs, separated by a space, to indicate the files that are to be added to the archive. After these properties are set, the .Action property is set to ETCP_ACTION_ADD to create the new archive and add the file. If the archive already exists, the file is added to the existing archive.

To extract a file from an archive, the .FileSpec property is set to the name of the archive to extract the file from. The .ZipFile property is set to the name of the file to extract. Once these properties have been set, setting the .Action property to ETCP_ACTION_EXTRACT extracts the file from the archive.

Features

The Basic Constituent control provides a basic set of features for creating and managing archives. The reference contains some sections whose text is grayed out. These items are not available in this edition of the control, but are left to demonstrate the full functionality of the **EITech Development's Compression Plus** version of the control.

This section outlines some of the features of the Zip constituent. Only items available in the control (not grayed out) are listed.

Create an Archive and add a file to the archive

To create a new archive and add a file, set the .ZipFile property to the name of the archive and the .FileSpec property to the name of the file to be added. Then set the .Action property to ETCP_ACTION_ADD.

Extract a file from an archive

To extract a file from an archive, set the .ZipFile property to the name of the archive and the .FileSpec property to the name of the file to extract. Then set the .Action property to ETCP_ACTION_EXTRACT.

Add a comment to an archive

The Comment property allows you to define a comment for the archive. The comment you add may be up to 1024 bytes long. The comment is added to the file specified in the FileName property.

Determine the size of a file in an archive

The CompSize property retrieves the compressed size for the file specified in the FileName property. The OrigSize property retrieves the original size of the file.

Retrieve the date for a file in the Archive

The Date property contains the date for the file specified in the .FileName property. The date is returned in the default system format specified in the international section of the WIN.INI file.

Set the location for a file after it is extracted from an archive

The DestPath property sets the specifies the target drive and/or path where files are placed when extracted from an archive. When DestPath is set to Null, the files are placed in the current directory. DestPath is Null by default.

Determine the number of files contained within an archive

The .FileCount property specifies the number of entries in the archive specified by the ZipFile property.

Determine if a file in the archive is encrypted

The Encrypted property is a boolean that indicates whether or not the file specified in the .FileName property is encrypted.

See the reference section for more information on these and other features of the Zip constituent.

Zip Control Reference

This control was provided by EllTech Development. It is a limited functionality edition of EllTech Development's full powered control (Compression Plus).

Note: Text that is strike through means that the feature is only available in the full release version of Compression Plus by EllTech Development. The documentation has been left in its original context to maintain consistency between the two versions and to also show the differences between the constituent control and the full feature version.

Note: Standard properties (Index, Left, Tag, etc.) are described in the Visual Basic manual and are not documented in this chapter.

The following format is used to document each of the bbbZip properties.

Property Name	Property Description
----------------------	-----------------------------

When the property is available as more than one operation, this section will be subdivided into appropriate categories as follows.

Zip: This section defines how the property can be used for zip operations.

Unzip: This section defines how the property can be used for unzip operations.

Dir: This section defines how the property can be used for directory services.

If the property is available as only one type of operation, the subheadings above will not be used.

Usage [form.]ctlname.Property = string\$
string\$ = [form.]ctlname.Property

This section shows the syntax for proper usage of this property.

Data Type String This section lists the property's data type.

Availability Design-Time Run-Time Read Run-Time Write

This section identifies the property's availability to the programmer.

See Also Property2, Event1

This section lists other properties or events that are related to this property.

Properties (Zip Constituent)

Description

Methods

Events

Property Name

Property Description

Action

The Action property is available as a Zip operation, UnZip operation, and Directory Service. When using the Action property, set all related properties first. When the Action property is set, the specified action is executed immediately.

Zip: You can use the Action property for zip operations when you are not using the QZip property. Set the Action property to one of the following values when creating and updating archives.

Setting	Value	Description
ETCP_ACTION_ADD	2	Adds all files that match the specifications defined in the FileSpec property. The files are added to the archive specified in ZipFile.
ETCP_ACTION_DELETE	5	Deletes all files from the archive that match the specifications defined in FileSpec.
ETCP_ACTION_FRESHEN	4	Updates files in an archive that have a later date/time stamp without adding files to the archive.
ETCP_ACTION_UPDATE	3	Updates files in an archive that have a later date/time stamp. Files that match the specifications defined in FileSpec are also added to the archive.
ETCP_ACTION_NONE	0	No actions taken.

Unzip: You can use the Action property for unzip operations when you are not using the QUnZip property. Set the Action property to one of the following values when extracting files from an archive.

Setting	Value	Description
ETCP_ACTION_TEST	7	Tests the integrity of an archive without extracting the files.
ETCP_ACTION_EXTRACT	6	Extracts files from an archive.

Dir: The Action property can be used to interrogate an archive file. First set the ZipFile property to the name of the Zip archive to

interrogate. Then, set the Action property to ETCP_ACTION_EXAMINE or 1.

During an ETCP_ACTION_EXAMINE operation, the Zip file remains opened and locked. Set Action to ETCP_ACTION_NONE or 0 when you are finished examining the archive directory.

Usage [form.]ctlname.Action = integer%
integer% = [form.]ctlname.Action

Data Type Integer

See Also FileSpec, QUnZip, QZip, ZipFile

AfterDate

~~Zip: The AfterDate property is used when searching for files to add to an archive. The AfterDate property specifies a date (month, day, and year format as defined by the sShortDate setting in the International section of the WIN.INI file) used to limit the file search. All files with a date/time stamp greater than or equal to the date specified in AfterDate will be added to the archive.~~

~~Usage [form.]ctlname.AfterDate = string\$
string\$ = [form.]ctlname.AfterDate~~

~~Data Type String~~

~~Availability Design-Time Run-Time Read Run-Time Write~~

~~See Also BeforeDate~~

Attributes

Dir: The Attributes property contains the file's DOS attributes (read-only, etc) of the current entry being examined. To determine which attributes are set, simply perform a logical AND of this value with the appropriate constant from the following table. More than one attribute bit may be set.

Setting	Value	Description
ETCP_ATTR_NORMAL	0	Normal file attribute
ETCP_ATTR_READONLY	1	Read-only file attribute
ETCP_ATTR_HIDDEN	2	Hidden file attribute
ETCP_ATTR_SYSTEM	4	System file attribute
ETCP_ATTR_VOLUME	8	Disk volume label
ETCP_ATTR_DIRECTORY	16	Subdirectory
ETCP_ATTR_ARCHIVE	32	Archive

The following code segment demonstrates how you can use the Attributes property:

```
If BbbZip1.Attributes AND ETCP_ATTR_ARCHIVE  
Then  
    Print "Archive"  
End If
```

Usage integer% = [form.]ctlname.Attributes

Data Type Integer

Availability Design-Time Run-Time Read Run-Time Write

BeforeDate

See Also FileArchiveBit, IncludeHidSys, ReadOnly
Zip: The BeforeDate property is used when searching for files to add to an archive. The BeforeDate property specifies a date (month, day, and year format as defined by the sShortDate setting in the International section of the WIN.INI file) used to limit the file search. All files with a date/time stamp earlier than the date specified in BeforeDate will be added to the archive.

Usage [form.]ctlname.BeforeDate = string\$
string\$ = [form.]ctlname.BeforeDate

Data Type String

Availability Design-Time Run-Time Read Run-Time Write

Cancel

See Also AfterDate
The Cancel property is available as both a Zip and Unzip property. Cancel has no effect when no operation is in progress.

Zip: The Cancel property cancels the current Zip operation. Set Cancel to True to cancel the current operation.

Unzip: The Cancel property cancels the current Unzip operation. Set Cancel to True to cancel the current operation.

Usage [form.]ctlname.Cancel = boolean%

Data Type Boolean

Availability Design-Time Run-Time Read Run-Time Write

Comment

See Also Action
Dir: The Comment property defines the comment for the archive entry specified in the FileName property. The Comment property can be used to add or modify a file's comment. The maximum length for a file comment is 1024 bytes. The ASCII NULL character is not allowed.

Usage [form.]ctlname.Comment = string\$
string\$ = [form.]ctlname.Comment

Data Type String

Availability Design-Time Run-Time Read Run-Time Write

See Also FileName, ZipComment

CompMethod

Zip: The CompMethod property specifies the compression method and speed used during Zip operations. The following settings are valid for the CompMethod property:

Setting	Value	Description
ETCP_METHOD_NONE	0	Store uncompressed.
ETCP_METHOD_DEFLATE0	1	Deflate fastest: fastest compression speed, low compression efficiency.
ETCP_METHOD_DEFLATE1	2	Deflate fast: medium compression speed, medium compression efficiency.
ETCP_METHOD_DEFLATE2	3	Default compression method: balance between compression speed and efficiency.
ETCP_METHOD_DEFLATE3	4	Deflate maximum compression: slow compression speed, high compression efficiency.
ETCP_METHOD_SCRUNCH	5	Use the Scrunch method, a variant of the LZ78 algorithm: fast compression, good efficiency.
ETCP_METHOD_IMPLODE	6	Use the Implode method, a combination of the Huffman and LZ77 algorithms: slow compression, high efficiency.
ETCP_METHOD_HUFFMAN	7	Use the Huffman method, a variant of the classic Huffman algorithm: medium compression speed, medium efficiency.
ETCP_METHOD_MASH	8	Use the Mash method, a combination of the LZ and Huffman methods: fast compression, good efficiency.
ETCP_METHOD_SHRINK	9	Use the Shrink method, a variant of the LZW algorithm: fast compression, medium efficiency.

ETCP_METHOD_SCRUNCH, ETCP_METHOD_HUFFMAN, and ETCP_METHOD_MASH are not compatible with PkZip.

Usage ~~[form.]ctlname.CompMethod = integer%~~
~~integer% = [form.]ctlname.CompMethod~~

Data Type ~~Integer (Enumerated)~~

Availability ~~Design-Time Run-Time Read Run-Time Write~~

CompSize

See Also ~~ZipFile~~

Dir: The CompSize property retrieves the compressed size for the file specified in the FileName property.

Usage long& = [form.]ctlname.CompSize

Data Type Long Integer

Availability Design-Time Run-Time Read Run-Time Write

CRC

See Also FileName, OrigSize

Dir: The CRC property retrieves the 32-bit CRC value for the file specified in the FileName property.

Usage long& = [form.]ctlname.CRC

Data Type Long Integer

Availability Design-Time Run-Time Read Run-Time Write

CreateDirs
{True}

See Also FileName

Unzip: The CreateDirs property can be used to automatically recreate entire directory hierarchies when restoring archived files. When CreateDirs is set to True, and the entries in the ZIP file specified by ZipFile were stored with relative pathnames, the subdirectory hierarchy will automatically be created on the target drive. CreateDirs is set to False by default.

Usage ~~[form.]ctlname.CreateDirs = boolean%~~
~~boolean% = [form.]ctlname.CreateDirs~~

Data Type ~~Integer~~

Availability ~~Design-Time Run-Time Read Run-Time Write~~

Date

See Also ~~ZipFile~~

Dir: The Date property retrieves the file date for the file specified in the FileName property. The date is returned in the default system format specified in the sShortDate setting in International section of the WIN.INI file.

Usage string\$ = [form.]ctlname.Date

Data Type String

Availability Design-Time Run-Time Read Run-Time Write

DestPath
See Also FileName, Time
Unzip: The DestPath property specifies the target drive and/or path where files are placed when extracted from an archive. When DestPath is set to Null, the files are placed in the current directory. DestPath is Null by default.

Usage [form.]ctlname.DestPath = string\$
 string\$ = [form.]ctlname.DestPath

Data Type String

Availability Design-Time Run-Time Read Run-Time Write

DirIndex
See Also CreateDirs
Dir: The DirIndex property specifies the index number of the current archive directory entry. Most of the Directory Services properties contain data reflecting different information associated with this entry. The valid values for DirIndex range from 0 to FileCount - 1. Note than an archive can contain entries other than files. Valid entries include files, subdirectory names, and volume labels. Use the Attributes property to determine the type of entry at a specific index. The EntryTypes property can be used to limit the types of entries returned.

Usage [form.]ctlname.DirIndex = long&
 long& = [form.]ctlname.DirIndex

Data Type Long Integer

Availability Design-Time Run-Time Read Run-Time Write

Encrypted
See Also Attributes, EntryTypes, FileCount
Dir: The Encrypted property indicates whether or not the file specified in the FileName property is encrypted. True is returned when the file is encrypted.

Usage boolean% = [form.]ctlname.Encrypted

Data Type Boolean

Availability Design-Time Run-Time Read Run-Time Write

EntryTypes
See Also FileName
Dir: The EntryTypes property specifies the type of archive entries that are included when an archive is examined. The following values are valid for the EntryTypes property.

Setting	Value	Description
ETCP_ENTRIES_FILES	0	(Default) Include only files.

ETCP_ENTRIES_DIRVOL	1	Include only directory and volume labels.
ETCP_ENTRIES_ALL	2	Include all entries in the archive.

The Attributes property can be used to determine the type of entry being examined. Simply AND the Attributes property with ETCP_ATTR_DIRECTORY for directories or ETCP_ATTR_VOLUME for volume labels. All others are files.

Usage [form.]ctlname.EntryTypes = integer%
integer% = [form.]ctlname.EntryTypes

Data Type Enumerated Integer

Availability Design-Time Run-Time Read Run-Time Write

See Also Attributes

ExcludeSpec The ExcludeSpec property is available during both Zip and Unzip operations.

Zip: The ExcludeSpec property defines the filename(s) or filespec(s) that will be unconditionally excluded from the current zip operation. For example, if FileSpec="*.*" and ExcludeSpec="*.BMP", all files would be processed except for files with a .BMP extension.

Unzip: The ExcludeSpec property defines the filename(s) or filespec(s) that will be unconditionally excluded from the current unzip operation. For example, if FileSpec="*.*" and ExcludeSpec="*.BMP", all files would be processed except for files with a .BMP extension.

Usage [form.]ctlname.ExcludeSpec = string\$
string\$ = [form.]ctlname.ExcludeSpec

Data Type String

Availability Design-Time Run-Time Read Run-Time Write

See Also FileSpec

ExtractFreshen Unzip: The ExtractFreshen property works with the FileSpec and ExcludeSpec properties to define additional restrictions on unzip operations. The ExtractFreshen property restricts unzip operations based on whether the file in the archive is newer or whether the file already exists in the directory specified by DestPath. The following settings are valid for the ExtractFreshen property:

Setting	Value	Description
ETCP_EXFR_ALL	0	Unconditionally extracts the files.
ETCP_EXFR_NEWEREXISTS	1	Extracts a file only if the version in the archive is newer and the file already

exists in the specified path.

ETCP_EXFT_NEWER 2 Extracts a file if the file in the archive is newer or the file does not already exist in the specified path.

Usage [form.]jetlname.ExtractFreshen = integer%
integer% = [form.]jetlname.ExtractFreshen

Data Type Integer (Enumerated)

Availability Design-Time Run-Time Read Run-Time Write

See Also ExcludeSpec, FileSpec

FileArchiveBit

Zip: The FileArchiveBit property specifies the method by which bbbZip handles files that have the DOS archive bit set. The following settings are valid for the FileArchiveBit property:

Setting	Value	Description
ETCP_FAB_IGNORE	0	Ignore the archive bit.
ETCP_FAB_ONLYIFSET	1	Add only files that have the archive bit set.
ETCP_FAB_ALLRESET	2	Add any file. Turn off the archive bit if set.
ETCP_FAB_ONLYIFSETRESET	3	Add only files with the archive bit set. Turn off the archive bit after adding the file.

Usage [form.]jetlname.FileArchiveBit = integer%
integer% = [form.]jetlname.FileArchiveBit

Data Type Integer (Enumerated)

Availability Design-Time Run-Time Read Run-Time Write

See Also IncludeHidSys

FileCount

Dir: The FileCount property specifies the number of entries in the archive specified by the ZipFile property.

Usage long& = [form.]jetlname.FileCount

Data Type Long Integer

Availability Design-Time Run-Time Read Run-Time Write

See Also EntryTypes, ZipFile

FileName

Dir: The FileName property retrieves the path (if stored) and name of a file, subdirectory name, or volume label. Since an archive may contain

different types of entries, the Attributes property should be used to determine the entry type. To determine the entry type, simply AND the Attributes property with ETCP_ATTR_DIRECTORY for directories or ETCP_ATTR_VOLUME for volume labels. All others are files.

Usage string\$ = [form.]ctlname.FileName

Data Type String

Availability Design-Time Run-Time Read Run-Time Write

See Also Attributes, EntryTypes

FileSpec

The FileSpec property is available during Zip and Unzip operations.

Zip: The FileSpec property specifies the filename(s) or filespec(s) to add, update, freshen, or delete to/from an archive. Each filename/spec must be separated by a space.

Unzip: The FileSpec property specifies the filename(s) or filespec(s) to test or extract from an archive. Each filename/spec must be separated by a space.

Usage [form.]ctlname.FileSpec = string\$
string\$ = [form.]ctlname.FileSpec

Data Type String

Availability Design-Time Run-Time Read Run-Time Write

See Also ExcludeSpec, ExtractFreshen

IncludeHidSys

Zip: The IncludeHidSys property defines the way bbbZip handles files that have either the hidden or the system attribute set. The following settings are valid for the IncludeHidSys property:-

Setting	Value	Description
ETCP_HIDSYS_EXCLUDE	0	(Default) Ignore files with either the system or the hidden attribute set.
ETCP_HIDSYS_HIDDEN	1	Include hidden files.
ETCP_HIDSYS_SYSTEM	2	Include system files.
ETCP_HIDSYS_BOTH	3	Include both hidden and system files.

Usage [form.]ctlname.IncludeHidSys = integer%
integer% = [form.]ctlname.IncludeHidSys

Data Type Integer (Enumerated)

Availability Design-Time Run-Time Read Run-Time Write

LongFileNames

See Also [FileArchiveBit](#)

Zip: The LongFileNames property defines the way bbbZip handles files that have long file name components.

Usage [form.]ctlname.LongFileNames = integer%
integer% = [form.]ctlname.LongFileNames

Data Type Boolean

Availability Design-Time Run-Time Read Run-Time Write

Method

Dir: The Method property contains the current file compression method. The following values are valid for the Method property:

Setting	Value	Description
ETCP_METHOD_NONE	0	Store uncompressed.
ETCP_METHOD_DEFLATE0	1	Deflate fastest: fastest compression speed, low compression efficiency.
ETCP_METHOD_DEFLATE1	2	Deflate fast: medium compression speed, medium compression efficiency.
ETCP_METHOD_DEFLATE2	3	Default compression method: balance between compression speed and efficiency.
ETCP_METHOD_DEFLATE3	4	Deflate maximum compression: slow compression speed, high compression efficiency.
ETCP_METHOD_SCRUNCH	5	Use the Scrunch method, a variant of the LZ78 algorithm: fast compression, good efficiency.
ETCP_METHOD_IMPLODE	6	Use the Implode method, a combination of the Huffman and LZ77 algorithms: slow compression, high efficiency.
ETCP_METHOD_HUFFMAN	7	Use the Huffman method, a variant of the classic Huffman algorithm: medium compression speed, medium efficiency.
ETCP_METHOD_MASH	8	Use the Mash method, a combination of the LZ and Huffman methods: fast

compression, good efficiency.

ETCP_METHOD_SHRINK 9 Use the Shrink method, a variant of the LZW algorithm: fast compression, medium efficiency.

ETCP_METHOD_SCRUNCH, ETCP_METHOD_HUFFMAN, and ETCP_METHOD_MASH are not compatible with PkZip.

Usage integer% = [form.]ctlname.Method

Data Type Integer

Availability Design-Time Run-Time Read Run-Time Write

See Also CompMethod

MoveFiles

Zip: The MoveFiles property specifies whether or not files will be removed from the source drive as they are added, updated, or refreshed in an archive. Set MoveFiles to True to remove files from the source drive after zip operations. MoveFiles is False by default. The MoveFiles property is equivalent to the PKZIP -m switch. Files are only deleted when the operation is successful.

Usage [form.]ctlname.MoveFiles = boolean%
boolean% = [form.]ctlname.MoveFiles

Data Type Boolean

Availability Design-Time Run-Time Read Run-Time Write

See Also FileSpec

MultiFirst

Zip: The MultiFirst property specifies the size of the first segment in a multi-volume archive. When creating a multi-volume archive, you may want to first create the files on the hard disk, sized according to the target media (for example a 1.44 MB floppy). The first disk in a multi-volume archive usually contains other data such as the SETUP program and the README file. Therefore, the archive size on the first disk is usually smaller than the capacity of the disk. The MultiFirst property allows you to specify a specific size for the first segment in the multi-volume archive. When MultiFirst is set to zero, all available disk space is used on the target drive. This value must be set a nonzero value. The minimum value is 4096.

Usage [form.]ctlname.MultiFirst = long&
long& = [form.]ctlname.MultiFirst

Data Type Long Integer

Availability Design-Time Run-Time Read Run-Time Write

MultiRest

See Also ——— MultiRest, MultiVolume

The MultiRest property specifies the size of the second and subsequent segments in a multi-volume archive. When creating a multi-volume archive, you may want to first create the files on the hard disk, sized according to the target media (for example a 1.44 MB floppy). The MultiFirst property allows you to specify a specific size for the first segment in the archive. The MultiRest property is used to specify the size for the remaining segments in the multi-volume archive. When MultiRest is set to zero, all available disk space is used on the target drive. When this value is not set to zero, it must be set to a value greater than or equal to 4096.

Usage — [form.]ctlname.MultiRest = long&
—— long& = [form.]ctlname.MultiRest

Data Type ——— Long Integer

Availability ——— Design-Time ——— Run-Time Read ——— Run-Time Write

MultiVolume

See Also ——— MultiFirst, MultiVolume, MultiVolumeType

The MultiVolume property is available during Zip and Unzip operations.

Zip: Use the MultiVolumeType property to set the type of multi-volume archive to create. When creating a multi-volume archive file, set the MultiVolume property to True. The file specified in ZipFile must not yet exist. bbbZip does not support the updating of multi-volume archives. The MultiVolume property is equivalent to the PKZIP -& switch. MultiVolume is False by default.

Unzip: When extracting files from a multi-volume archive file, set the MultiVolume property to True. bbbZip can extract files from multi-volume archive files created with bbbZip, PkZip, and DynaZip.

Usage — [form.]ctlname.MultiVolume = boolean%
—— boolean% = [form.]ctlname.MultiVolume

Data Type ——— Boolean

Availability ——— Design-Time ——— Run-Time Read ——— Run-Time Write

MultiVolumeType

See Also ——— MultiFirst, MultiRest, MultiVolumeType

Zip: The MultiVolumeType property specifies the type of multi-volume archive to create. The default Compression Plus format is the best choice since this format allows you to create multi-volume archives on fixed media. The MultiFirst and MultiRest properties are used to determine the size of the first and subsequent multi-volume files, respectively. The following settings are valid for the MultiVolumeType property:

Setting	Description
0	(Default) Use the Compression Plus format.
1	Use the PKZIP format.
2	Use the Inner Media DynaZip format.

Usage ~~[form.]ctlname.MultiVolumeType = integer%~~
~~integer% = [form.]ctlname.MultiVolumeType~~

Data Type ~~Integer~~

Availability ~~Design-Time Run-Time Read Run-Time Write~~

OrigSize

See Also ~~MultiFirst, MultiRest~~

Dir: The OrigSize property contains the original uncompressed size of the file specified by the FileName property.

Usage long& = [form.]ctlname.OrigSize

Data Type Long Integer

Availability Design-Time Run-Time Read Run-Time Write

OverWrite

See Also CompSize, FileName

Unzip: The OverWrite property defines the way bbbZip reacts when it encounters a file that already exists in the destination subdirectory specified by DestPath. The following values are valid for the OverWrite property.

Setting	Value	Description
ETCP_OVERWRITE_PROMPT	0	Prompts the user to confirm the overwrite operation. The Prompt event is triggered when the Prompts property is True.
ETCP_OVERWRITE_ALWAYS	1	Overwrites the file without prompting.
ETCP_OVERWRITE_NEVER	2	Does not overwrite existing files.

Usage [form.]ctlname.OverWrite = integer%
integer% = [form.]ctlname.OverWrite

Data Type Integer (Enumerated)

Availability Design-Time Run-Time Read Run-Time Write

Password

See Also DestPath, Prompt event, Prompts

The Password property is available during Zip and Unzip operations.

Zip: The Password property allows you to encrypt files with a password as they are added, updated, or freshened in an archive. The password is a string with length up to 64 characters. Password is empty by default. This property is equivalent to the PKZIP -s[password] switch. Do not forget your password! If you do, you will not be able to

recover your files.

Unzip: The Password property allows you to extract encrypted files from an archive. To extract an encrypted file, simply set the Password property to the same password that was used to encrypt the file. If you do not have the correct password, you will not be able to extract the file.

Do not use ASCII Character 34 (the " mark) or ASCII Character 0 in the Password property.

Usage [form.]ctlname.Password = string\$
string\$ = [form.]ctlname.Password

Data Type String

Availability Design-Time Run-Time Read Run-Time Write

See Also FileSpee

Prompts

The Prompts property is available during Zip and Unzip operations.

Zip: The Prompts property indicates the way bbbZip handles control events and user prompts. When Prompts is True, bbbZip fires Progress and Prompt control events when appropriate. When Prompts is False, no prompts or progress information is displayed. Prompts is True by default.

Unzip: The Prompts property indicates the way bbbZip handles control events and user prompts. When Prompts is True, bbbZip fires Progress and Prompt control events when appropriate. When Prompts is False, no prompts or progress information is displayed. Prompts is True by default.

Whenever a Prompt event is triggered, bbbZip displays a standard, built-in dialog by default. You can add custom code to the Prompt event to handle some or all of the possible types of prompts. When Prompts is False, the user will not be prompted for overwrite confirmation and bbbZip will not yield any timeslices to other Windows applications. If OverWrite is 0, bbbZip will skip the file without overwriting.

Usage [form.]ctlname.Prompts = boolean%
boolean% = [form.]ctlname.Prompts

Data Type Boolean

Availability Design-Time Run-Time Read Run-Time Write

See Also Progress event, Prompt event

QUnZip

Unzip: The QUnZip property is equivalent to the PKUNZIP command line. With QUnZip, you can carry out a command-line unzip operation while ignoring all other unzip properties (with the exception of ReadOnly and Prompts). See Appendix C for a complete list of Compression Plus unzip switches. Keep in mind that all other bbbZip

~~unzip properties are ignored when QUnZip is used (again, with the exception of ReadOnly).~~

~~Usage [form.]ctlname.QUnZip = string\$~~

~~Data Type String~~

~~Availability Design-Time Run-Time Read Run-Time Write~~

~~See Also Zip, ReadOnly~~

QZip

~~Zip: The QZip property is equivalent to the PKZIP command line. With QZip, you can carry out a command-line zip operation while ignoring all other zip properties (with the exception of Prompts). See Appendix B for a complete list of Compression Plus zip switches. Keep in mind that all other bbbZip zip properties are ignored when QZip is used. QZip cannot be used to create multi-volume archives.~~

~~Usage [form.]ctlname.QZip = string\$~~

~~Data Type String~~

~~Availability Design-Time Run-Time Read Run-Time Write~~

~~See Also QUnZip~~

ReadOnly

~~Unzip: The ReadOnly property specifies the way bbbZip opens an archive. Set ReadOnly to True to open an archive with READ_ONLY access. This property is useful when opening an archive from a CD-ROM drive or network drive where write-access is not granted.~~

~~Usage [form.]ctlname.ReadOnly = boolean%
boolean% = [form.]ctlname.ReadOnly~~

~~Data Type Boolean~~

~~Availability Design-Time Run-Time Read Run-Time Write~~

~~See Also QUnZip~~

Recurse

~~Zip: The Recurse property is equivalent to the PKZIP -r switch. When Recurse is True, all subdirectories beneath the directory level as indicated by the FileSpec property are also searched for matching files. Recurse is False by default.~~

~~Usage [form.]ctlname.Recurse = boolean%
boolean% = [form.]ctlname.Recurse~~

~~Data Type Boolean~~

~~Availability Design-Time Run-Time Read Run-Time Write~~

~~See Also FileSpec, StorePaths~~

Status

~~The Status property is available during Zip, Unzip, and Directory operations.~~

Zip: The Status property indicates the current status of zip operation. A return value of 0 indicates no error. See Appendix A for a list of error codes.

Unzip: The Status property indicates the current status of an unzip operation. A return value of 0 indicates no error. See Appendix A for a list of error codes.

Dir: The Status property indicates the current status of a directory operation. A return value of 0 indicates no error. See Appendix A for a list of error codes

Usage integer% = [form.]ctlname.Status

Data Type Integer

Availability Design-Time Run-Time Read Run-Time Write

See Also Action, Cancel

StorePaths

Zip: The StorePaths property indicates whether pathnames are stored in the archive along with the filenames. The following values are valid for the StorePaths property:

Setting	Value	Description
ETCP_PATHS_NONE	0	No pathnames are stored.
ETCP_PATHS_RECURSED	1	If Recurse is True, all relative paths are stored in the archive.
ETCP_PATHS_SPECIFIEDRECURSED	2	If Recurse is True, all relative paths are stored in the archive.

Paths specified in FileSpec are also stored regardless of Recurse.

Usage [form.]ctlname.StorePaths = integer%
integer% = [form.]ctlname.StorePaths

Data Type Integer (Enumerated)

Availability Design-Time Run-Time Read Run-Time Write

See Also FileSpec, Recurse

Time

Dir: The Time property returns the time of the file specified by the FileName property. The file time is returned in the default system format as defined in the International section of the WIN.INI file.

Usage string\$ = [form.]ctlname.Time

UseVolLabel

Data Type String

Availability Design-Time Run-Time Read Run-Time Write

See Also Date, FileName

The UseVolLabel property is available during Zip and Unzip operations.

Zip: This property allows you to store a volume label from a specified drive into the ZIP archive during a zip operation. Set this property to the drive letter of the desired drive before the zip operation. Setting this property to the <SPACE> character indicates the default drive. When more than one character is specified, the first character is used as the drive letter. Only one volume label can be stored in an archive. This property is equivalent to the PKZIP switch -\$.

Unzip: Set this property to any non-null value to add or set the current volume label on the drive specified by the DestPath property. A volume label must have been previously stored in the archive.

Usage [form.]ctlname.UseVolLabel = string\$
 string\$ = [form.]ctlname.UseVolLabel

ZipComment

Data Type String

Availability Design-Time Run-Time Read Run-Time Write

See Also DestPath

Dir: The ZipComment property specifies the comment for the archive specified in the ZipFile property. Use this property to add or modify the archive comment. The maximum length of the comment is 2048 bytes. The comment cannot include ASCII character 0.

Usage [form.]ctlname.ZipComment = string\$
 string\$ = [form.]ctlname.ZipComment

ZipFile

Data Type String

Availability Design-Time Run-Time Read Run-Time Write

See Also Comment, ZipFile

The ZipFile property is available as a Zip operation, Unzip operation, and Directory Service. The ZipFile property must be defined before setting the Action property to any of Action's predefined constants (with the exception of ETCP_ACTION_NONE). If you modify this property while an action is in progress, the action will be canceled.

Zip: The ZipFile property specifies the name of the archive file to create or update.

Unzip: The ZipFile property specifies the name of the archive containing the files to be extracted or tested.

Dir: The ZipFile property specifies the archive file to be interrogated.

Usage [form.]ctlname.ZipFile = string\$
string\$ = [form.]ctlname.ZipFile

Data Type String

Availability Design-Time Run-Time Read Run-Time Write

See Also FileName, ZipFileDate

ZipFileDate

Zip: The ZipFileDate property specifies the way the date and time stamp of an archive is to be modified after a zip operation (add, update, or freshen). The following values are valid for the ZipFileDate property:

Setting	Value	Description
ETCP_ZIPDATE_CURRENT	0	(Default) The archive is stamped with the date and time of the modification.
ETCP_ZIPDATE_ORIGINAL	1	The archive is stamped with the original date and time.
ETCP_ZIPDATE_LATEST	2	The archive is stamped with the date and time of the newest file in the archive.

Usage [form.]ctlname.ZipFileDate = integer%
integer% = [form.]ctlname.ZipFileDate

Data Type Enumerated Integer

Availability Design-Time Run-Time Read Run-Time Write

See Also ZipFile

Methods (Zip Constituent)

Description

Properties

Events

There are no methods for this control.

Events (Zip Constituent)

<u>Description</u>	<u>Properties</u>	<u>Methods</u>
Event Name	Description	
Progress	The Progress event is triggered periodically as each file is compressed or decompressed. This event procedure is the ideal location for handling progress reports and status displays. The Progress event procedure is defined as follows:	

Sub Progress (Operation%, MsgType%, Percent%, Method%, ZipFile\$, FileName\$)

Operation	Value	Description
ETCP_ACTION_NONE	0	No action.
ETCP_ACTION_EXAMINE	1	Directory services
ETCP_ACTION_ADD	2	Add files to an archive
ETCP_ACTION_UPDATE	3	Update an archive
ETCP_ACTION_FRESHEN	4	Freshen an archive
ETCP_ACTION_DELETE	5	Delete files from an archive
ETCP_ACTION_EXTRACT	6	Extract files from an archive
ETCP_ACTION_TEST	7	Test archive files without extracting

A MsgType value of zero indicates that no error occurred and the event was fired to allow you to display a standard progress report.

A non-zero MsgType value indicates an error condition or that the current Action was canceled by the user. The following is a list of the possible values for MsgType.

Note that additional MsgType values may be listed in the include files which define these constants, as the include files may be updated more frequently than this manual. See the files for the latest information.

Value	Description
ETCP_ERR_NOT_ENOUGH_MEMORY	Not enough memory
ETCP_ERR_INVALID_HANDLE	Invalid (file or archive) handle
ETCP_ERR_ACCESS_DENIED	Access denied (file error)
ETCP_ERR_PATH_NOT_FOUND	Path not found (or unable to create subdirectory)
ETCP_ERR_FILE_NOT_FOUND	File not found
ETCP_ERR_NO_PROBLEM	No error
ETCP_ERR_UNKNOWN_ARCHIVE_TYPE	Not a ZIP archive
ETCP_ERR_INVALID_ARCHIVE_DIR	Error in archive directory
ETCP_ERR_UNKNOWN_METHOD	Unknown compression method
ETCP_ERR_USER_ABORT	User wants to cancel
ETCP_ERR_NO_FILES_FOUND	No matching files
ETCP_ERR_INCOMPLETE	Unable to complete request
ETCP_ERR_INVALID_SWITCH	Invalid switch option
ETCP_ERR_DISK_FULL	Write error (disk full?)
ETCP_ERR_READ_PAST_END	Read error (read past end?)
ETCP_ERR_INVALID_OFFSET	Invalid starting offset for EtUnzipString, EtUnzipMem, or EtUnzipMemPtr
ETCP_ERR_OVERFLOW	Overflow (integer > 32767)

ETCP_ERR_NOTHING_TO_DO	No action specified [respond by displaying help]
ETCP_ERR_SWITCH_CONFLICT	Conflicting switches were specified
ETCP_ERR_TOO_MANY_FILES	Too many files to process (more than 65535)
ETCP_ERR_INVALID_DATE	Invalid date specified
ETCP_ERR_BAD_FILE_MODE	Invalid file mode specified (e.g., tried to write to a read-only file)
ETCP_ERR_BAD_PASSWORD	Incorrect password
ETCP_ERR_BAD_CRC	Computed CRC doesn't match recorded CRC
ETCP_ERR_NOT_WITH_MV	Function is not supported for multivolume archives
ETCP_ERR_CANT_CONTINUE	ZIP was modified between Et4ZipFindNext calls
ETCP_ERR_NOT_REMOVABLE	Fixed disk was not specified for a MV archive format that requires removable media

The Percent parameter indicates how much of the current file, specified by FileName, has been processed. This information can easily be applied to a Percent Complete control or something similar.

The Method parameter specifies the compression method that was/is being applied to the current file. The following values are valid for the Method parameter.

Setting	Value	Description
ETCP_METHOD_NONE	0	Store uncompressed.
ETCP_METHOD_DEFLATE0	1	Deflate fastest: fastest compression speed, low compression efficiency.
ETCP_METHOD_DEFLATE1	2	Deflate fast: medium compression speed, medium compression efficiency.
ETCP_METHOD_DEFLATE2	3	Default compression method: balance between compression speed and efficiency.
ETCP_METHOD_DEFLATE3	4	Deflate maximum compression: slow compression speed, high compression efficiency.
ETCP_METHOD_SCRUNCH	5	Use the Scrunch method, a variant of the LZ78 algorithm: fast compression, good efficiency.
ETCP_METHOD_IMPLODE	6	Use the Implode method, a combination of the Huffman and LZ77 algorithms: slow compression, high efficiency.
ETCP_METHOD_HUFFMAN	7	Use the Huffman method, a variant of the classic Huffman algorithm: medium compression speed, medium efficiency.
ETCP_METHOD_MASH	8	Use the Mash method, a combination of the LZ and Huffman methods: fast compression, good efficiency.
ETCP_METHOD_SHRINK	9	Use the Shrink method, a variant of the

LZW algorithm: fast compression,
medium efficiency.

The ZipFile parameter specifies the name of the current archive.

If you wish to cancel the current operation, set the Cancel property to True prior to exiting the Progress event procedure.

The Progress event is also the ideal place to allocate time slices to other applications. A DoEvents statement placed just before exiting the event procedure should work well.

Prompt The Prompt event controls the way bbbZip handles user prompts. When the Prompts property is True (the default), the Prompt event is triggered whenever a decision is required by the end-user. This includes asking for file overwrite confirmation, asking for the next disk in a multivolume archive disk sequence, etc. The procedure definition for the Prompt event follows:

```
Sub bbbZip1_Prompt (Operation%, MsgType%,  
DiskNum%, ZipFile$, FileName$, Reply%)
```

When the Prompt event is triggered, the Operation parameter will be set to the current Action. Operation is one of the following values:

<u>Operation</u>	<u>Value</u>	<u>Description</u>
ETCP_ACTION_NONE	0	No action.
ETCP_ACTION_EXAMINE	1	Directory services
ETCP_ACTION_ADD	2	Add files to an archive
ETCP_ACTION_UPDATE	3	Update an archive
ETCP_ACTION_FRESHEN	4	Freshen an archive
ETCP_ACTION_DELETE	5	Delete files from an archive
ETCP_ACTION_EXTRACT	6	Extract files from an archive
ETCP_ACTION_TEST	7	Test archive files without extracting

You can examine the MsgType parameter to determine the type of prompt needed (overwrite confirmation, multivolume disk change, etc.). You can either display a dialog yourself, or instruct bbbZip to use its default dialogs. The following values are valid for the MsgType parameter:

<u>Value</u>	<u>Value</u>	<u>Description</u>
ETCP_QUERY_INSERT_DISK	1	Insert a specific disk # in a multivolume set
ETCP_QUERY_OVERWRITE	2	Overwrite confirmation
ETCP_QUERY_INSERT_LAST	3	Insert the last disk in a PK-compatible multivolume set

On entry, the Reply parameter is set to ETCP_YOU_DO_IT (-10,000). If Reply is unchanged on exit, no action was taken during the event and Compression Plus should use its own default dialog box for this message.

The DiskNum parameter contains the number of the next disk required by the multivolume backup/restore sequence when MsgType is set to ETCP_QUERY_INSERT_DISK.

The ZipFile parameter contains the name of the current archive.

The FileName parameter contains the name of the current file being added, extracted, etc.

On exit from this event procedure, Reply should be set to indicate how Compression Plus should proceed. The actual value to use varies according to the message. The following list contains the possible messages and possible return values for Reply.

MsgType	Description
ETCP_QUERY_OVERWRITE	Asks for file overwrite confirmation. FileName contains the name of the file to be overwritten.
Reply Values	
IDYES - Overwrite	
IDNO - Do NOT overwrite	
IDCANCEL - Abort unzip operation	
ETCP_QUERY_INSERT_DISK	Asks for a specific disk number during a multi-volume operation. DiskNum contains the number of the requested disk.
Reply Values	
IDOK - Disk inserted. Continue.	
IDCANCEL - Cancel the multivolume operation	
ETCP_QUERY_INSERT_LAST	Asks for the last disk in a multivolume set to be inserted. This message is possible only when working with PKZIP-compatible multivolume archives.
Reply Values	
IDOK - Disk inserted. Continue.	
IDCANCEL - Cancel the multivolume operation	

IDYES, IDNO, IDCANCEL, and IDOK are defined in the Visual Basic file CONSTANT.TXT. These constants are also defined in the Windows SDK Help file and the WINDOWS.H file provided with the Microsoft Windows SDK. You can also use the defined VB constants vbYES, vbNO, vbCANCEL, vbOK.

