See also

FontName Property Array
FontBold Property Array
FontItalic Property Array
FontSize Property Array
FontStrikethru Property Array
FontUnderline Property Array
Message Property Array

See also

AdjustFieldHeight Property
FontName Property Array
FontSize Property Array
Message Property Array

See also

AdjustFieldHeight Property
FontBold Property Array
FontItalic Property Array
FontSize Property Array
FontStrikethru Property Array
FontUnderline Property Array
Message Property Array

See also

AdjustFieldHeight Property
FontBold Property Array
FontItalic Property Array
FontName Property Array
FontStrikethru Property Array
FontUnderline Property Array
Message Property Array

See also

AutoToggle Property
FieldType Property Array
FieldWidth Property Array
KeyOffText, KeyOnText Properties
Message Property Array
Toggle Event

See also

AutoToggle Property
FieldType Property Array
FieldWidth Property Array
KeyOffText, KeyOnText Properties
Message Property Array
Numlock, ScrollLock, Capslock Properties

See also

FieldType Property Array
FieldWidth Property Array
KeyOffText, KeyOnText Properties
Message Property Array
Numlock, ScrollLock, Capslock Properties
Toggle Event

See also

FieldWidth Property Array

Message Property Array

See also

FieldWidth Property Array

See also

FieldType Property Array

FieldWidth Property Array

See also

BackColor Property Array
FieldWidth Property Array
FloodColor Property
FloodField Property
FloodPercent Property
FloodShowPct Property
ForeColor Property Array
Message Property Array

See also

BackColor Property Array
FieldWidth Property Array
FloodColor Property
FloodField Property
FloodInvertText Property
FloodPercent Property
ForeColor Property Array
Message Property Array

See also

BackColor Property Array
FieldWidth Property Array
FloodColor Property
FloodField Property
FloodInvertText Property
FloodShowPct Property
ForeColor Property Array
Message Property Array

See also

BackColor Property Array
FloodColor Property
FloodInvertText Property
FloodPercent Property
FloodShowPct Property
ForeColor Property Array
Message Property Array

See also

BackColor Property Array
FloodField Property
FloodInvertText Property
FloodPercent Property
FloodShowPct Property
ForeColor Property Array
Message Property Array

See also

FieldWidth Property Array

See also

Alignment Property Array
BackColor Property Array
ForeColor Property Array
Message Property Array

See also

Alignment Property Array
BackColor Property Array
FieldWidth Property Array
ForeColor Property Array

See also

FieldWidth Property Array
Message Property Array
SysMenuBrowse Event
Tag Property

See also

FieldWidth Property Array
Message Property Array
MenuTagsField Property
Tag Property

See also

BackColor Property Array
FieldWidth Property Array
RedrawField Property
Message Property Array

See also

FieldWidth Property Array
ForeColor Property Array
RedrawField Property
Message Property Array

See also

FieldOutline Property Array
FieldType Property Array
SpaceAfter Property Array

See also

FieldType Property Array
FieldWidth Property Array
Message Property Array

See also

FieldType Property Array
FieldWidth Property Array
Message Property Array

See also

AutoToggle Property
FieldOutline Property Array
FieldWidth Property Array
KeyOffText, KeyOnText Properties
Message Property Array
Toggle Event

See also

FieldOutline Property Array
FieldWidth Property Array

See also

AutoToggle Property
FieldType Property Array
Numlock, ScrollLock, Capslock Properties
Toggle Event

Toggle Example, StatusBar Control

Copy

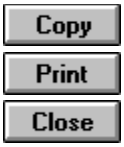
Print

Close

```
Const FIELD_NORMAL = 0
Const FIELD_CAPSLOCK = 1
Const FIELD_NUMLOCK = 2
Const FIELD_SCROLLOCK = 3
Const FIELD_DATETIME = 4

Sub Form_Load ()

    Const ALIGN_LEFT = 0
    Const ALIGN_RIGHT = 1
    Const ALIGN_CENTER = 2

    Const OFF = 0
    Const ON  = 1

    ' Align the statusbar to the bottom of the window
    ' and set the height
    StatusBar1.Align = 2
    StatusBar1.Height = 330

    ' Set messages for the toggle-keys
    StatusBar1.CapsLockOnText = "CAPS"
    StatusBar1.CapsLockOffText = "caps"
    StatusBar1.NumLockOnText = "NUM"
    StatusBar1.NumLockOffText = "num"
    StatusBar1.ScrollLockOnText = "SCROLL"
    StatusBar1.ScrollLockOffText = "scroll"

    ' The statusbar initializes three fields by setting their
    ' FieldWidth property to a value greater than zero.
    StatusBar1.FieldWidth(0) = 400
    StatusBar1.FieldWidth(1) = 40
    StatusBar1.FieldWidth(2) = 40
    StatusBar1.FieldWidth(3) = 40

    ' The spacing between the fields is set
    StatusBar1.SpaceAfter(0) = 4
    StatusBar1.SpaceAfter(1) = 2
    StatusBar1.SpaceAfter(2) = 2
    ' SpaceAfter(3) isn't set since this is the last field
    ' and it's SpaceAfter property isn't used.

    StatusBar1.FieldType(0) = FIELD_NORMAL    ' Plain text (DEFAULT)
    StatusBar1.FieldType(1) = FIELD_NUMLOCK   ' Num-lock key status
    StatusBar1.FieldType(2) = FIELD_CAPSLOCK  ' Caps-lock key status
    StatusBar1.FieldType(3) = FIELD_SCROLLOCK ' Scroll-lock key status

    ' Allow the user to double-click a toggle-field to toggle the key
    StatusBar1.AutoToggle = True
```

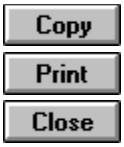
```
    ' Turn num-lock on and turn caps- and scroll-lock off.
    StatusBar1.NumLock = ON
    StatusBar1.CapsLock = OFF
    StatusBar1.ScrollLock = OFF

    ' The alignment for the last three fields is set to ALIGN_CENTER
    StatusBar1.Alignment(1) = ALIGN_CENTER    ' Center
    StatusBar1.Alignment(2) = ALIGN_CENTER    ' Center
    StatusBar1.Alignment(3) = ALIGN_CENTER    ' Center

    ' Set a message for the first field (the other three
    ' messages are ignored anyhow
    StatusBar1.Message(0) = "Demonstration of the StatusBar control"

End Sub

Sub StatusBar1_Toggle(FieldType As Integer, KeyState As Integer)
Dim sMsg As String
    Select Case FieldType
          Case FIELD_NUMLOCK: sMsg = "Numlock"
          Case FIELD_CAPSLOCK: sMsg = "Capslock"
          Case FIELD_SCROLLOCK: sMsg = "Scroll-lock"
    End Select
    sMsg = sMsg + " is turned "
    If KeyState Then sMsg = sMsg + "on."  Else sMsg = sMsg + "off."
    MsgBox sMsg
End Sub
```

Progress Indicator Example, StatusBar Control




```
Sub Form_Load ()

    ' Only display one field of the statusbar control
    StatusBar1.FieldWidth(0)=100
    ' Set the field to automatically expand
    StatusBar1.ExpandField = 0
    ' Set the field to show a percentage
    StatusBar1.FloodField = 0
    ' Set the field to invert the text not under the filled area
    StatusBar1.FloodInvertText = True
    ' Don't display the percentage
    StatusBar1.FloodShowPct = False

    ' Set a message for the percentage field
    StatusBar1.Message(0) = "Busy creating database..."
    StatusBar1.Align = 2
    StatusBar1.Height = 400
End Sub

Sub Form_Click ()
    StatusBar1.FloodPercent = StatusBar1.FloodPercent + 5
End Sub
```

Click Example, StatusBar Control




```
Sub Form_Load ()
Dim I As Integer

    WindowState = 2
    StatusBar1.Align = 2
    StatusBar1.Height = 400

    For I = 0 To 7
          StatusBar1.FieldWidth(I) = 50
          StatusBar1.Message(I) = "Field " & Format$(I)
    Next I

    StatusBar1.FieldWidth(8) = 200
    StatusBar1.Message(8) = "Click on any field"
End Sub

Sub StatusBar1_Click(Field As Integer)
Dim sMsg As String
    If Field<>8 Then
          sMsg = "Clicked on field " & Format$(Field)
          StatusBar1.ForeColor(8) = QBColor(0)
    Else
          sMsg = "Hey, you clicked on me!"
          StatusBar1.ForeColor(8) = RGB(0,0,255)
    End If
    StatusBar1.Message(8) = sMsg
End Sub
```

StatusBar Initialize Example, StatusBar Control

Copy

Print

Close

```
Sub Form_Load ()

    Const FIELD_NORMAL = 0
    Const FIELD_CAPSLOCK = 1
    Const FIELD_NUMLOCK = 2
    Const FIELD_SCROLLOCK = 3
    Const FIELD_DATETIME = 4

    Const ALIGN_LEFT = 0
    Const ALIGN_RIGHT = 1
    Const ALIGN_CENTER = 2

    Const RAISED = 1
    Const HEAVY_RAISED = 2
    Const INSET = 3
    Const HEAVY_INSET = 4

    Const OUTLINE_NONE = 0
    Const OUTLINE_RAISED = 1
    Const OUTLINE_INSET = 2

    ' Align the statusbar to the bottom of the window
    ' and set the height
    StatusBar1.Align = 2
    StatusBar1.Height = 330

    ' Make the statusbar display a slightly inset font
    StatusBar1.Font3D = INSET

    ' The statusbar initializes three fields by setting their
    ' FieldWidth property to a value greater than zero.
    StatusBar1.FieldWidth(0) = 400
    StatusBar1.FieldWidth(1) = 33
    StatusBar1.FieldWidth(2) = 70

    ' The first field is set to AutoExpand
    ' Since field 0 is the expand field, the specified width
    ' will be treated as the minimum width for this field"
    StatusBar1.ExpandField = 0

    ' All fields get a different outline
    StatusBar1.FieldOutline(0) = FIELD_NONE   ' no outline
    StatusBar1.FieldOutline(1) = FIELD_RAISED ' raised outline
    StatusBar1.FieldOutline(2) = FIELD_INSET  ' inset outline

    ' All fields receive their own Forecolor
    StatusBar1.ForeColor(0) = QBColor(12)     ' Red
    StatusBar1.ForeColor(1) = QBColor(0)      ' Black
    StatusBar1.ForeColor(2) = RGB(0,0,255)    ' Bright Blue
```

```
    ' Field 0 gets a different backcolor
    StatusBar1.BackColor(0) = QBColor(1)      ' Dark Blue

    ' The spacing between the fields is set
    StatusBar1.SpaceAfter(0) = 4
    StatusBar1.SpaceAfter(1) = 2
    ' SpaceAfter(2) isn't set since this is the last field
    ' and it's SpaceAfter property isn't used.

    StatusBar1.FieldType(0) = FIELD_NORMAL    ' Plain text (DEFAULT)
    StatusBar1.FieldType(1) = FIELD_NUMLOCK   ' Num-lock key status
    StatusBar1.FieldType(2) = FIELD_DATETIME  ' Date-time
    StatusBar1.DateTimeFormat(2) = "General Date"

    StatusBar1.FieldOutline(0) = OUTLINE_NONE       ' No outline for field 0
    StatusBar1.FieldOutline(1) = OUTLINE_RAISED     ' Raised outline
    StatusBar1.FieldOutline(2) = OUTLINE_INSET      ' Inset outline

    ' The alignment for the last two field is set to ALIGN_CENTER
    StatusBar1.Alignment(1) = ALIGN_CENTER    ' Center
    StatusBar1.Alignment(2) = ALIGN_CENTER    ' Center

    ' Set a message for the first field (the other two
    ' messages are ignored anyhow
    StatusBar1.Message(0) = "Demonstration of the StatusBar control"

End Sub
```

Toggle Event, StatusBar Control

see also example

Description

Occurs when the user presses one of the Caps-lock, Num-lock or Scroll-lock keys on the keyboard or double-click a field width FieldType 1, 2 or 3 and the AutoToggle property has been set to True.

Syntax

Sub *StatusBar1*_**Toggle** (*Index* **As Integer**, *FieldType* **As Integer,** *KeyState* **As Integer**)

Remarks

The argument *Index* uniquely identifies a control if it is in a control array. The *FieldType* argument specifies the special key that was toggled. It has one of the following three values:

Value	Short name	Description
1	FIELD_CAPSLOCK	Caps-lock has been toggled
2	FIELD_NUMLOCK	Num-lock has been toggled
3	FIELD_SCROLLOCK	Scroll-lock has been toggled

The *KeyState* argument is a boolean argument and specifies whether the toggled key was toggled on or off.

The example shows you a possible way to react to the Toggle event. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

SysMenuBrowse Event, StatusBar Control

see also

Description

Occurs when the user selects an item from the system-menu on the form the StatusBar is on.

Syntax

Sub *StatusBar1*_**SysMenuBrowse** (*MenuID* **As Integer**, *Message* **As String**)

Remarks

This event allows you to specify a message whenever a user browses through the items in the system menu. It only occurs if the MenuTagsField property of the StatusBar control is set to something other than -1.

You can specify a message to be displayed, according to the menuID parameter, by modifying the Message parameter. The menuID parameter specifies the id of the currently selected menu-item of the system-menu. It is one of the following:

MenuID	Menu-item
> 0	System menu box
- 4096	Size
- 4080	Move
- 4064	Minimize
- 4048	Maximize
- 4032	Next window
- 4000	Close
- 3808	Restore
- 3792	Switch to ..

If the text in the Message parameter is not empty, it is displayed in the field specified by the MenuTagsField property.

DblClick Event, StatusBar Control

see also

Description

Occurs when the user quickly presses and then releases the left-mouse button over a field of the StatusBar control twice. You can not trigger the DblClick event for the StatusBar control in code.

Syntax

Sub *StatusBar1*_**DblClick** (*Index* **As Integer**, *Field* **As Integer**)

Remarks

The argument *Index* uniquely identifies a control if it is in a control array. The *Field* argument specifies the number of the text-field that was doubleclicked on. You can use this number to take some action whenever the user doubleclicks on a field.

Click Event, StatusBar Control

see also example

Description

Occurs when the user presses and then releases the left-mouse button over a field of the StatusBar control. You can not trigger the Click event for the StatusBar control in code.

Syntax

Sub *StatusBar1*_**Click** (*Index* **As Integer**, *Field* **As Integer**)

Remarks

The argument *Index* uniquely identifies a control if it is in a control array. The *Field* argument specifies the number of the text-field that was clicked on. You can use this number to take some action whenever the user clicks on a field.

The example shows you a possible way to react to the Click event. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

FontSize Property Array, StatusBar Control

see also

Description

Determines the size of the font to be used for a text-field of the StatusBar control.

Usage

[*form!*]*StatusBar1*.**FontSize(I)** [= *setting*]

Remarks

Use this font property to specify the size of the text displayed in the text-field of StatusBar control.

Note

Fonts available in Visual Basic vary according to your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which actual fonts exist.In general, you should change the FontName property before you set size and style attributes with the FontSize, FontBold, FontItalic, FontStrikethru, and FontUnderline properties.

However, when you set TrueType fonts to smaller than 8 points, you should set the point size with the FontSize property, then set the FontName property, and then set the size again with the FontSize property. The Windows environment uses a different font for TrueType fonts that are smaller than 8 points.

The index in the FontSize property array is the same index as in the FieldWidth and Message arrays.

Data Type

Single

FontBold, FontItalic, FontStrikethru and FontUnderline Property Arrays, StatusBar Control

see also

Description

Determine font styles for the text in the text-field in the following formats: **FontBold**, *FontItalic*, ~~FontStrikethru~~, and FontUnderline.

Usage

[*form!*]*StatusBar1*.**FontBold(I)** [= *setting%*]

[*form!*]*StatusBar1*.**FontItalic(I)** [= *setting%*]

[*form!*]*StatusBar1*.**FontStrikethru(I)** [= *setting%*]

[*form!*]*StatusBar1*.**FontUnderline(I)** [= *setting%*]

Remarks

The settings for the FontBold, FontItalic, FontStrikethru, and FontUnderline properties are:

Setting	Description
True	Turns on the formatting in that style.
False	(default) Turns off the formatting in that style.

Use these font properties to format the look of the text displayed in text-field of the StatusBar control.

Note

Fonts available in Visual Basic vary according to your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which actual fonts exist.In general, you should change the FontName property before you set size and style attributes with the FontSize, FontBold, FontItalic, FontStrikethru, and FontUnderline properties.

However, when you set TrueType fonts to smaller than 8 points, you should set the point size with the FontSize property, then set the FontName property, and then set the size again with the FontSize property. The Windows environment uses a different font for TrueType fonts that are smaller than 8 points.

The index in these property arrays are the same indexes as in the FieldWidth and Message arrays.

Data Type

Integer (Boolean)

FontName Property Array, StatusBar Control

see also

Description

Determines the font used to display the messages for a text-field.

Usage

[*form!*]*StatusBar1*.**FontName(I)** [= *setting$*]

Remarks

The default for this property is determined by the system. Fonts available with Visual Basic vary according to your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which fonts exist.

The index in the FontName property array is the same index as in the FieldWidth and Message arrays.

In general, you should change FontName before setting size and style attributes with the FontSize, FontBold, FontItalic, FontStrikethru and FontUnderline properties.

On systems running Windows 3.0, the fonts "Helv" or "Tms Rmn" are called "MS Sans Serif" and "MS Serif", respectively. In code, if you set FontName to "Helv," then test whether the FontName is set to "Helv," the result will be False, since it will be changed internally to "MS Sans Serif."

Data Type

String (Boolean)

Font3D Property, StatusBar Control

see also example

Description

Specifies how that StatusBar control displays the text in the text-fields.

Usage

[*form!*]*StatusBar1*.**Font3D** [= *setting%*]

Remarks

The StatusBar can display fonts in five different ways. The Font3D property takes one of the following values to specify the way the Statusbar displays the font in a text-field.

Value	Short name	Description
0	DEFAULT	(Default) Simply displays the selected font.
1	RAISED	Displays the font slightly raised.
2	HEAVY_RAISED	Displays a heavy raised font.
3	INSET	Displays the font slightly inset.
4	HEAVY_INSET	Display a heavy inset font.

Data Type

Integer (Enumerated)

The example shows a Form_Load that initializes three text-fields on a StatusBar control. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

DateTimeFormat Property Array, StatusBar Control

see also example

Description

Specifies the date or time-format for text-fields with FieldType 4 (FIELD_DATETIME).

Usage

[*form!*]*StatusBar1*.**DateTimeFormat(I)** [= *setting$*]

Remarks

To format dates and times, you can use either the commonly used formats that have been predefined in Visual Basic or create user-defined time formats using standard characters that have special meaning when used in a format expression.

The following table shows the predefined data format names you can use and the meaning of each:

Format Name	Description
General Date	Display a date and time (e.g. 4/3/93 05:34 PM).
Long Date	Display a Long Date, as defined in the International section of the Control Panel.
Medium Date	Display a date in the same form as the Short Date, as defined in the International section of the Control Panel, except spell out the month abbreviation.
Short Date	Display a Short Date, as defined in the International section of the Control Panel.
Long Time	Display a Long Time, as defined in the International section of the Control Panel. Long Time includes hours, minutes, seconds.
Medium Time	Display time in 12-hour format using hours and minutes and the AM/PM designator.
Short Time	Display a time using the 24-hour format (e.g. 17:45)

The following table shows the characters you can use to create user-defined date/time formats and the meaning of each:

Character	Meaning
c	Display the date as ddddd and display the time as t t t t t, in that order. Only date information is displayed if there is no fractional part to the date serial number; only time information is displayed if there is no integer portion.
d	Display the day as a number without a leading zero (1-31).
dd	Display the day as a number with a leading zero (01-31).
ddd	Display the day as an abbreviation (Sun-Sat).
dddd	Display the day as a full name (Sunday-Saturday).
ddddd	Display a date serial number as a complete date (including day, month, and year) formatted according to the Short Date setting in the International section of

	the Windows Control Panel. The default Short Date format is m/d/yy.
dddddd	Display a date serial number as a complete date (including day, month, and year) formatted according to the Long Date setting in the International section of the Control Panel. The default Long Date format is mmmm dd, yyyy.
w	Display the day of the week as a number (1 for Sunday through 7 for Saturday.)
ww	Display the week of the year as a number (1-53).
m	Display the month as a number without a leading zero (1-12). If m immediately follows h or hh, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01-12). If m immediately follows h or hh, the minute rather than the month is displayed.
mmm	Display the month as an abbreviation (Jan-Dec).
mmmm	Display the month as a full month name (January-December).
q	Display the quarter of the year as a number (1-4).
y	Display the day of the year as a number (1-366).
yy	Display the year as a two-digit number (00-99).
yyyy	Display the year as a four-digit number (100-9999).
h	Display the hour as a number without leading zeros (0-23).
hh	Display the hour as a number with leading zeros (00-23).
n	Display the minute as a number without leading zeros (0-59).
nn	Display the minute as a number with leading zeros (00-59).
s	Display the second as a number without leading zeros (0-59).
ss	Display the second as a number with leading zeros (00-59).
t t t t t	Display a time serial number as a complete time (including hour, minute, and second) formatted using the time separator defined by theTime Format in the International section of the Control Panel. A leading zero is displayed if the Leading Zero option is selected and the time is before 10:00 A.M. or P.M. The default time format is h:mm:ss.
AM/PM	Use the 12-hour clock and display an uppercase AM with any hour before noon; display an uppercase PM with any hour between noon and 11:59 PM.
am/pm	Use the 12-hour clock and display a lowercase AM with any hour before noon; display a lowercase PM with any hour between noon and 11:59 PM.
A/P	Use the 12-hour clock and display an uppercase A with any hour before noon; display an uppercase P with any hour between noon and 11:59 PM.
a/p	Use the 12-hour clock and display a lowercase A with any hour before noon; display a lowercase P with any hour between noon and 11:59 PM.
AMPM	Use the 12-hour clock and display the contents of the

1159 string (s1159) in the WIN.INI file with any hour before noon; display the contents of the 2359 string (s2359) with any hour between noon and 11:59 PM. AMPM can be either uppercase or lowercase, but the case of the string displayed matches the string as it exists in the WIN.INI file. The default format is AM/PM.

The following are examples of user-defined date and time formats:

Format	Display
m/d/yy	12/7/58
d-mmmm-yy	7-December-58
d-mmmm	7 December
mmmm-yy	December 58
hh:mm AM/PM	08:50 PM
h:mm:ss a/p	8:50:35 p
h:mm	20:50
h:mm:ss	20:50:35
m/d/yy h:mm	12/7/58 20:50

Data Type

String

The example shows a Form_Load that initializes three text-fields on a StatusBar control. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

NumLock, CapsLock and ScrollLock Properties, StatusBar Control

see also example

Description

Sets or returns the current state of the num-lock, caps-lock or scroll-lock key.

Usage

[*form!*]*StatusBar1*.**NumLock** [= *setting%*]

[*form!*]*StatusBar1*.**CapsLock** [= *setting%*]

[*form!*]*StatusBar1*.**ScrollLock** [= *setting%*]

Remarks

You can use the NumLock, CapsLock and ScrollLock properties to set or get the state of the num-lock, caps-lock keys.

The NumLock, CapsLock and ScrollLock properties are only available at run-time, since they are of no use in the design time environment (its much quicker to press one of the keys than to set it explicitly through the Properties-window). These properties always reflect the current state of the toggle keys, so if you set the NumLock property to one (ON) and a user presses the Num-Lock key after that, getting the NumLock property will return zero (OFF). There are two possible settings for the NumLock, CapsLock and ScrollLock properties:

Value	Short Name	Meaning
0	OFF	The state of num--lock, caps-lock or scroll-lock is (turned) off.
1	ON	The state of num--lock, caps-lock or scroll-lock is (turned) on.

Data Type

Integer (Enumerated)

The example demonstrates the use of the NumLock, CapsLock and ScrollLock properties. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

BackColor Property Array, StatusBar Control

see also example

Description

Sets or retrieves the field back-color of one of the text-fields of the StatusBar control.

Usage

[*form!*]*StatusBar1*.**BackColor(I)** [= *color&*]

Remarks

Visual Basic uses the Microsoft Windows environment RGB scheme for colors. Each property has the following ranges of settings:

Range of settings	Description
Normal RGB colors	Colors by using the RGB or QBColor functions in code.
System default colors	Colors specified with system color constants from CONSTANT.TXT, a Visual Basic file that specifies system defaults. The Windows environment substitutes the user's choices as specified in the user's Control Panel settings.

For the StatusBar control the default settings at design time are:

BackColor(I) set to BUTTON_FACE system color as specified in CONSTANT.TXT.

Setting the BackColor property does not affect messages already displayed to avoid flickering if you want to display a new message in a different color. To force the current message to be redisplayed in the new color set the RedrawField property to the number of the field to redisplay.

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower three bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte is not 0, Visual Basic uses the system colors, as defined in the user's Control Panel and enumerated in CONSTANT.TXT.

Data Type

Long

The example shows a Form_Load that initializes three text-fields on a StatusBar control. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

AutoToggle Property, StatusBar Control

see also example

Description

Specifies if the user can toggle the special keys num-lock, caps-lock and scroll-lock by double clicking on a field with FieldType 1, 2 or 3.

Usage

[*form!*]*StatusBar1*.**AutoToggle** [= *setting%*]

Remarks

The AutoToggle property only has effect when you have defined fields with FieldType 1, 2 or 3. There are two possible settings for the AutoToggle property:

AutoToggle	Result
True	Toggles num-lock, caps-lock or scroll-lock when a user double-clicks on such a field.
False	Does not toggle num-lock, caps-lock or scroll-lock on double-clicks.

Data Type

Integer (Boolean)

The example demonstrates the use of the AutoToggle property. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

AdjustFieldHeight Property, StatusBar Control

see also

Description

Specifies if the height of the text-field should depend on the currently selected font for that text-field or should be set to the height of the text-field with the largest font.

Usage

[*form!*]*StatusBar1*.**AdjustFieldHeight** [= *setting%*]

Remarks

The AdjustFieldHeight allows you to have fields with different heights on the StatusBar depending on the size of the Font that has been specified for that field. If you've specified for example a FontSize of 10 for field 0 and a FontSize of 14 for field 1, setting the AdjustFieldHeight property to FALSE will cause all fields to have the same height. If you set AdjustFieldHeight to TRUE, will make the height of field 0 smaller than the height of Field 1.

There are two possible settings for the AdjustFieldHeight property:

AdjustFieldHeight	Result
True	The height of each text-field depends on the size of the font specified for that field. If fonts are different among text-fields, the heights of the fields will not always be the same.
False	(Default) The height of each text-field depends on the size of the largest font specified in all text-fields. This will ensure an equal height for all text-fields.

Data Type

Integer (Boolean)

FloodInvertText Property, StatusBar Control

see also example

Description

Specifies whether or not to change the color of the text according to the progress of the percentage in the text of the text-field specified by the the FloodField property.

Usage

[*form!*]*StatusBar1*.**FloodInvertText** [= *setting%*]

Remarks

The FloodInvertText property affects the text that is displayed in the text-field specified by the FloodField property. There are two possible settings for the FloodInvertText property:

FloodInvertText	Color of text-field
True	The text-color of the text displayed in the text-field is different for the text on the progress indicator bar and the space next to the bar. The text-color in the space right next to the bar is set to the same color as the color specified in the FloodColor property. The text on the bar is drawn with the color specified in the ForeColor property array for that field.
False	The entire text is displayed in the color specified in the ForeColor property array.

Data Type

Integer (Boolean)

The example demonstrates the use of the FloodInvertText property. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

FloodShowPct Property, StatusBar Control

see also example

Description

Specifies whether or not to include the percentage in the text of the text-field specified by the the FloodField property.

Usage

[*form!*]*StatusBar1*.**FloodShowPct** [= *setting%*]

Remarks

The FloodShowPct property affects the text that is displayed in the text-field specified by the FloodField property. Another thing that affects the text is the Message property for that text-field. There are four possible combinations:

FloodShowPct	Message	Contents of text-field
True	some text	The text specified in the Message property-array followed by a colon, followed by the percentage (taken from the FloodPercent property).
True	empty	Only the percentage.
False	some text	The text specified in the Message property-array.
False	empty	empty

Data Type

Integer (Boolean)

The example demonstrates the use of the FloodShowPct property. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

FloodPercent Property, StatusBar Control

see also example

Description

Specifies the percentage of the text-field specified by the FloodField property to be filled.

Usage

[*form!*]*StatusBar1*.**FloodPercent** [= *setting%*]

Remarks

The FloodPercent property sets or retrieves the percentage of the text-field that will be filled with the color specified by the FloodColor property. If the FloodPercent property is set to a value smaller than zero or greater than one hundred it's value is automatically adjusted. This setting will have no effect is the FloodField property is set to -1 or the FloodField property is set to a field with a FieldWidth of zero.

Data Type

Integer

The example demonstrates the use of the FloodPercent property. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

FloodField Property, StatusBar Control

see also example

Description

Specifies which field is to be used as a progress indicator.

Usage

[*form!*]*StatusBar1*.**FloodField** [= *setting%*]

Remarks

The FloodField property allows you to use one field of the StatusBar control as a progress indicator. The message for this field (specified in the Message property array) will be placed before the percentage specified in the FloodPercent property. A colon is automatically inserted between the message and the percentage. If no message is specified for the text-field the colon is omitted. The FloodShowPct property specifies whether or not to append the actual percentage to the message specified for the FloodField.

If the FloodField property is set to a field that has a FieldWidth of zero, the FloodField is not displayed.

The field is filled with the color specified by the FloodColor property.

Data Type

Integer

The example demonstrates the use of the FloodPercent property. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

FloodColor Property, StatusBar Control

see also example

Description

Sets or retrieves the color used to paint the area inside a text-field when the statusbar is used as a percentage indicator (only when the FloodField property is setting is other than -1).

Usage

[*form!*]*StatusBar1*.**FloodColor** [= *color&*]

Remarks

The FloodColor property has the same range of settings as standard Visual Basic color settings.

Use this property with FloodField and FloodPercent to cause the StatusBar to display a colored percentage bar indicating the degree of completion of a task.

At design time you can set this property by entering a hexadecimal value in the Settings box or by clicking the three dots that appear at the right of the Settings box. Clicking this button displays a dialog box that allows you to select a FloodColor setting from the Visual Basic Color Palette.

Note

The FloodColor property defaults to bright blue: RGB (0, 0, 255). The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower three bytes, from least to most significant, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF).

Data Type

Long

The example demonstrates the use of the FloodPercent property. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

SpaceAfter Property Array, StatusBar Control

see also example

Description

Sets or retrieves the spacing between the text-fields StatusBar control.

Usage

[*form!*]*StatusBar1*.**SpaceAfter(I)** [= *setting%*]

Remarks

The SpaceAfter property array allows you to define the spacing between the text-fields in the StatusBar control. It specifies how much space (in pixels) to leave behind a text-field before the next text-field is shown. The index in the SpaceAfter array is the same index as in the FieldWidth and Message arrays. So if you set SpaceAfter(0) to 4, you define a spacing of 4 pixels after the first field. The default setting is 3 pixels, but you can set it to any value you like.

Data Type

Integer

The example shows a Form_Load that initializes three text-fields on a StatusBar control. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

RedrawField Property, StatusBar Control

see also

Description

Forces the StatusBar to redisplay one or all text-fields.

Usage

[*form!*]*StatusBar1*.**RedrawField** [= *setting%*]

Remarks

If you specify an alternate Alignment and ForeColor for a text-field, the field is not updated automatically. This is done on purpose to avoid flickering if you want to display a new message in a different color.

If you only want to change the forecolor or alignment of a field without changing the message, use the RedrawField property to redisplay the message with the new settings. Specifying -1 for the RedrawField property forces all messages to be redrawn. Of course, you can also force the StatusBar control to redraw all fields by using the Refresh method or by assigning the Message property of a field to itself (StatusBar1.Message(0) = StatusBar1.Message(0)), but using the RedrawField property is more efficient.

This property is not available at design time and write-only at run-time.

Data Type

Integer

Message Property Array, StatusBar Control

see also example

Description

Sets or retrieves the text to be displayed in one of the text-fields of the StatusBar control.

Usage

[*form!*]*StatusBar1*.**Message(I)** [= *message$*]

Remarks

Setting a new message for a text-field automatically redisplays the message, so be sure to set other properties, like Alignment or ForeColor, before you change the message. If the message is to big to fit in the field, the message is clipped.

Only text-fields with a FieldWidth bigger than zero are displayed. Fields with zero-length are not displayed regardless of the message you specify for them

Data Type

String

The example shows a Form_Load that initializes three text-fields on a StatusBar control. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

MenuTagsField Property, StatusBar Control

see also

Description

Specifies which field of the StatusBar should be used to display menu tags.

Usage

[*form!*]*StatusBar1*.**MenuTagsField** [= *setting%*]

Remarks

The StatusBar is capable of tracking menu selections on the form it is placed upon. Every time a menu-item is selected, the StatusBar obtains the Tag property from that menu-item and displays it in the textfield specified by the MenuTagsField property.

So, the only thing you have to do to make the StatusBar automatically display messages when a user is browsing menu's, is to specify a sensible message in the Tag property of all menu-items and set the MenuTagsField property to the number of the textfield you want to display these messages in.

To prevent the StatusBar from displaying the text in the Tag-property of a menu, set this property to -1 or set it to a field width FieldWidth set to zero.

To show a message whenever the user select an item from the system menu on your form, react to the SysMenuBrowse event of the StatusBar control.

Special considerations

Only one StatusBar on a form can display the menu tags. If you have more than one StatusBar on a form and both of them have a value for the MenuTagsField other than -1, the StatusBar control that is loaded first will display the menu-messages. If you'd like to switch between the two StatusBar controls set the MenuTagsField property of the first StatusBar to -1 and reset the MenuTagsField property of the second StatusBar (e.g. StatusBar2.MenuTagsField = StatusBar2.MenuTagsField).

When you use a StatusBar on a form that has it's MDIChild property set to True, the MenuTagsField property for that StatusBar will have no effect. Instead, menu-messages for menus on MDI-child forms will be displayed in the StatusBar of the MDI-parent form (MDIForm) if there is one.

Data Type

Integer

LeftMargin, RightMargin Properties, StatusBar Control

Description

Sets or retrieves the left of right-margin for the StatusBar control.

Usage

[*form!*]*StatusBar1*.**LeftMargin** [= *setting%*]

[*form!*]*StatusBar1*.**RightMargin** [= *setting%*]

Remarks

The LeftMargin property specifies how much room to leave before displaying the first text-field, the RightMargin property specifies the space to reserve after the last text-field.

The Left- and Right margin are always expressed in pixels, regardless of the ScaleMode of the StatusBar's parent.

Data Type

Integer

ForeColor Property Array, StatusBar Control

see also example

Description

Sets or retrieves the field text-color of one of the text-fields of the StatusBar control.

Usage

[*form!*]*StatusBar1*.**ForeColor(I)** [= *color&*]

Remarks

Visual Basic uses the Microsoft Windows environment RGB scheme for colors. Each property has the following ranges of settings:

Range of settings	Description
Normal RGB colors	Colors by using the RGB or QBColor functions in code.
System default colors	Colors specified with system color constants from CONSTANT.TXT, a Visual Basic file that specifies system defaults. The Windows environment substitutes the user's choices as specified in the user's Control Panel settings.

For the StatusBar control the default settings at design time are:

ForeColor(I) set to the WINDOW_TEXT system color as specified in CONSTANT.TXT.

Setting the ForeColor property does not affect messages already displayed to avoid flickering if you want to display a new message in a different color. To force the current message to be redisplayed in the new color set the RedrawField property to the number of the field to redisplay.

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower three bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte is not 0, Visual Basic uses the system colors, as defined in the user's Control Panel and enumerated in CONSTANT.TXT.

To display text in the Windows environment, the text-colors must be solid. If the text-colors you've selected are not displayed, the selected color may be dithered that is, comprised of up to three different-colored pixels. If you choose a dithered color for the text, the nearest solid color will be substituted.

Data Type

Long

The example shows a Form_Load that initializes three text-fields on a StatusBar control. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

FieldWidth Property Array, StatusBar Control

see also example

Description

Set or retrieves the field width of one of the text-fields of the StatusBar control.

Usage

[*form!*]*StatusBar1*.**FieldWidth(I)** [= *setting%*]

Remarks

The FieldWidth of a field is always expressed in pixels. A field that has a field-width of zero is not displayed, nor is it's SpaceAfter property used in calculating the relative distances between the text-fields.

Data Type

Integer

The example shows a Form_Load that initializes three text-fields on a StatusBar control. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

Alignment Property Array, StatusBar Control

see also example

Description

Specifies the alignment for a text-field of the StatusBar control.

Usage

[*form!*]*StatusBar1*.**Alignment(I)** = *Alignment%*

Remarks

The StatusBar can align each text-field in three ways. The Alignment property array takes one of the following values to specify the alignment of a text-field.

Value	Short name	Description
0	LEFT	(Default) Aligns the text at the left-edge of the text-field.
1	RIGHT	Aligns the text at the right-edge of the text-field.
2	CENTER	Centers the text in the text-field.

The index in the Alignment property array is the same index in the Message property array.

Data Type

Integer (Enumerated)

The example shows a Form_Load that initializes three text-fields on a StatusBar control. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

FieldType Property Array, StatusBar Control

see also example

Description

Sets or retrieves the field-type for the text-field with the same index.

Usage

[*form!*]*StatusBar1*.**FieldType(I)** [= *FieldType%*]

Remarks

The StatusBar supports five different field-types. The FieldType property array takes one of the following values to specify the type of a text-field.

Value	Short name	Description
0	FIELD_NORMAL	(Default) The text-field displays the programmer defined text.
1	FIELD_CAPSLOCK	Displays the current status of the CapsLock key. If the CapsLock key is toggled on, the text from the property CapsLockOffText is displayed, otherwise the CapsLockOnText is displayed.
2	FIELD_NUMLOCK	Displays the current status of the NumLock key.
3	FIELD_SCROLLOCK	Displays the current status of the ScrollLock key.
4	FIELD_DATETIME	Displays a date or time depending on the corresponding setting in the DateTimeFormat property array.

The index in the FieldType property array is the same index in the Message and FieldWidth property arrays.

Data Type

Integer (Enumerated)

The example shows a Form_Load that initializes three text-fields on a StatusBar control. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

Insert button

If you click this button, a new textfield is inserted before the currently selected field. If you already defined 20 fields, the last textfield is moved off the StatusBar. All fields, including the currently selected field are moved one place down, to accomodate space for the new field.

Delete button

If you click this button, the currently selected textfield is deleted. All fields after this field are moved one place up. If you already defined 20 fields, the last field will become available again.

Up button

This button (together with the down-button) allows you to exchange textfields. If you click this button, the currently selected textfield is exchanged with the textfield right before it, effectively moving it one place up.

Down button

This button (together with the up-button) allows you to exchange textfields. If you click this button, the currently selected textfield is exchanged with the textfield after it, effectively moving it one place down.

Backcolor selection

You can select a field's background color by clicking the 'Color'-button. A preview of the selected font-properties, forecolor and backcolor for the currenctly selected field are shown to the left of the Color-button.

Clicking the Color-button will pop-up a standard Windows common-dialog where you can select a color from the available set of colors.

The background color can also be set by specifying a value for the BackColor property array at run-time.

Font and Forecolor selection

You can select the font that is to displayed and the text-color by clicking the 'Font'-button. A preview of the selected font-properties and forecolor for the currenctly selected field are shown to the left and below the Font-button.

Clicking the Font-button will pop-up a standard Windows common-dialog where you can select all font-related properties.

You can also set the font-related properties by setting the appropriate entries in the FontName, FontSize, FontBold, FontItalic, FontStrikeThru and FontUnderline property arrays at run-time.

The text-color can also be set by specifying a value for the ForeColor property array at run-time.

Cancel button

To ignore the changes you've made for the settings for the text-fields of the StatusBar control, click the Cancel-button. The StatusBar will remain exactly the same as it was before you selected the FieldProperties property.

Ok button

To confirm the settings for the text-fields of the StatusBar control, click the Ok-button.

All settings you've specified will be applied to the StatusBar control. Now you can see directly, how your StatusBar will look when the program is run.

Message

Allows you to set a message that will be displayed in the currently selected text-field of the StatusBar control. If the message is to big to fit in the field, the message is clipped.

Only text-fields with a FieldWidth bigger than zero are displayed. Fields with zero-length are not displayed regardless of the message you specify for them.

You can also set this property by setting the appropriate entry in the Message property array at run-time.

FloodField Checkbox

The FloodField check-box allows you to use one field of the StatusBar control as a progress indicator. The message for this field (specified in the Message property array) will be placed before the percentage specified in the FloodPercent property. A colon is automatically inserted between the message and the percentage. If no message is specified for the text-field the colon is omitted. The FloodShowPct property specifies whether or not to append the actual percentage to the message specified for the FloodField.

If the FloodField check-box is check on a field that has a FieldWidth of zero, the FloodField is not displayed.

The field is filled with the color specified by the FloodColor property.

You can also set this property by specifying a value for the FloodField property at run-time.

ExpandField Checkbox

The effect of the checking ExpandField check-box is that all other field are positioned first and that after that the remaining space is filled by the currently selected text-field (starting at the correct position of course). If you don't want a text-field to be stretched, find the field with the ExpandField check-box checked and uncheck it or check the ExpandField check-box on a text-field with FieldWidth set to zero. The FieldWidth property for the text-field that the ExpandField points to, will be regarded as the minimum width for that text-field. If the StatusBar control isn't wide enough to display all fields, the fields will extend off the StatusBar control.

You can also set this property by specifying a value for the ExpandField property at run-time.

Space After

The SpaceAfter text-box array allows you to define the spacing between the text-fields in the StatusBar control. It specifies how much space (in pixels) to leave behind the currently selected text-field before the next text-field is shown. So if you set the value to 4, you define a spacing of 4 pixels after the currently selected field. The default setting is 3 pixels, but you can set it to any value you like.

You can also set this property by setting the appropriate entry in the SpaceAfter property array at run-time.

Field Size

You can enter the width of the currently selected field in this text-box. The FieldWidth of a text-field is always expressed in pixels. A field that has a field-width of zero is not displayed, nor is it's SpaceAfter property used in calculating the relative distances between the text-fields.

You can also set this property by setting the appropriate entry in the FieldWidth property array at run-time.

Alignment selection combo-box

Select the alignment (the way text is justified within a text-field) for the currently selected field. Possible values are:

Value	Description
Left	(Default) The text is left aligned in the text-field
Right	Aligns the text to the right of the text-field
Center	Centers the text in the text-field

You can also set this property by setting the appropriate entry in the Alignment property array at run-time.

Field Outline selection combo-box

Select the outline (the border around the text-field) for the currently selected field. Possible values are:

Value	Description
None	The StatusBar draws no border around the text-field
Raised	Draws a raised border around the text-field.
Inset	(Default) Draws an inset border around the text-field.

You can also set this property by setting the appropriate entry in the FieldOutline property array at run-time.

FieldType selection combo-box

Select the type of text-field for the currently selected field. Possible values are:

Value	Description
Normal Field	(Default) The text-field displays the text specfied in the Message property array.
CapsLock Field	Displays the current status of the CapsLock key. If the CapsLock key is toggled on, the text from the property CapsLockOffText is displayed, otherwise the CapsLockOnText is displayed.
NumLock Field	Displays the current status of the NumLock key.
ScrollLock Field	Displays the current status of the ScrollLock key.
Date/Time Field	Displays a date or time depending on the setting of the Date/Time format combo box or the string specified in the DateTimeFormat property array.

You can also set this property by setting the appropriate entry in the FieldType property array at run-time.

Date/Time Format combo-box

This combo-box lets you define the format that will be used for a Date/Time field. You can select a predefined setting or you can type in a customized string. Legal formats include all date/time format strings you can use with the Visual Basic Format$ function. Setting this property is equivalent to specifying a value for the DateTimeFormat property array at run-time.

Field Listbox

Choose the field you want to specify from this listbox. When you select a field from the list, it's settings are displayed in the combo-boxes and text-boxes to the right of the listbox.

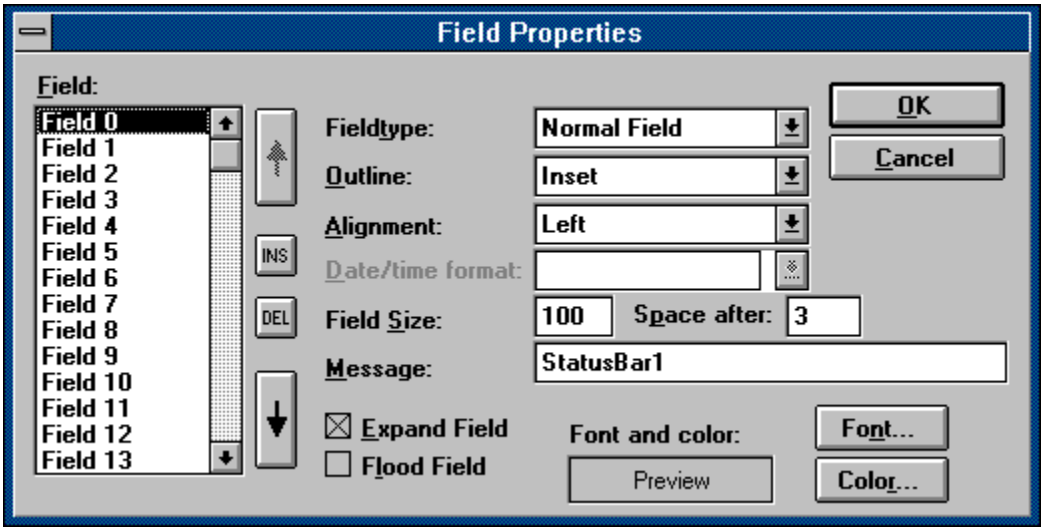
FieldProperties Dialog, StatusBar Control

Description

Allows you to set the properties for all text-fields on the StatusBar at design time, using a dialog-window.

Usage

Double click on the three dots next to the property to display the Field-Properties Dialog. The dialog looks like this. To get help on the way to use the dialog, click the various parts of it for detailed information.



FieldOutline Property Array, StatusBar Control

see also example

Description

Sets or retrieves the way the StatusBar draws the outline for the text-field with the same index.

Usage

[*form!*]*StatusBar1*.**FieldOutline(I)** [= *FieldOutline%*]

Remarks

The StatusBar supports three different field-outlines. The FieldType property array takes one of the following values to specify the outline that is drawn around the border of a text-field.

Value	Short name	Description
0	NONE	The StatusBar draws no border around the text-field
1	RAISED	Draws a raised border around the text-field.
2	INSET	(Default) Draws an inset border around the text-field.

The index in the FieldOutline property array is the same index in the Message and FieldWidth property arrays.

Data Type

Integer (Enumerated)

The example shows a Form_Load that initializes three text-fields on a StatusBar control. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

ExpandField Property, StatusBar Control

see also example

Description

The ExpandField property specifies which field is stretched to the size of the StatusBar control.

Usage

[*form!*]*StatusBar1*.**ExpandField** [= *setting%*]

Remarks

The effect of the ExpandField property is that all other field are positioned first and that after that the remaining space is filled by the text-field specified by the ExpandField property (starting at the correct position of course). The FieldWidth property for the text-field that the ExpandField points to, will be regarded as the minimum width for that text-field. If the StatusBar control isn't wide enough to display all fields, the fields will extend off the StatusBar control.

If you don't want any field stretched to the full size of your window, set the value of the ExpandField property to -1 or set the ExpandField property to a text-field with FieldWidth set to zero.

Data Type

Integer

The example shows a Form_Load that initializes three text-fields on a StatusBar control. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

CapsLockOffText, CapsLockOnText, NumLockOffText, NumLockOnText, ScrollLockOffText, ScrollLockOnText Properties, StatusBar Control

see also

Description

Specify the text to show for a text-field of type 2 (Caps-lock), 3 (Num-lock) or 4 (Scroll-lock).

Usage

[*form!*]*StatusBar1*.***Key*OnText** [= *setting$*]

[*form!*]*StatusBar1*.***Key*OffText** [= *setting$*]

Remarks

The *Key*OnText is shown when the specified key is toggled on, *Key*OffText is shown when the key is toggled off. You can specify the type of a text-field by setting the FieldType property.

Data Type

String

The example shows you a how these properties are used. To use this example create a form with one StatusBar control and paste the code into the Declarations section of your form.

Events

The StatusBar control supports the following events:

Click	DblClick	DragDrop	DragOver
SysMenuBrowse	Toggle		

Methods

The StatusBar control supports the following methods:

Drag	Move	Refresh	ZOrder

Properties

All the properties that apply to the Status Bar control are listed in the following table. All properties that are marked with an asterisk (*) are only available at run-time.

Align	AdjustFieldHeight	Alignment *	AutoToggle
BackColor *	CapsLock *	CapsLockOffText	CapsLockOnText
DateTimeFormat *	Enabled	ExpandField	FieldOutline *
FieldProperties	FieldType *	FieldWidth *	FloodColor
FloodField	FloodInvertText	FloodPercent	Font3D
FontBold *	FontItalic *	FontName *	FontSize *
FontStrikethru *	FontUnderline *	ForeColor *	Height
hWnd	Index	Left	LeftMargin
MenuTagsField	Message *	Name	NumLock *
NumLockOffText	NumLockOnText	Parent	RedrawField *
RightMargin	ScrollLock *	ScrollLockOffText	ScrollLockOnText
SpaceAfter *	Tag	Top	Visible
Width			

Note the Alignment, BackColor, CapsLock, DateTimeFormat, FontBold, FontName, FontItalic, FontSize, FontStrikethur, FontUnderline, FieldOutline, FieldType, FieldWidth, ForeColor, NumLock, RedrawField and ScrollLock properties are only available at run-time. These properties can also be set at design time using the FieldProperties dialog-box. Name is the default property for the StatusBar control.

The StatusBar Custom Control

Properties Methods Events

Description

The Microsoft Visual Basic programming system for Windows comes with a large set of 3D controls. Unfortunately, a 3D status-bar, as used in almost every MS-Windows application is lacking. Therefore the the Status Bar Custom Control was designed. This control allows you to create a very versatile status bar for all your applications.

File Name

TOOLBARS.VBX

Object Type

StatusBar

Toolbox Icon

Remarks

The StatusBar allows an application to display a status-bar at the bottom of a form. The statusbar has 20 fully configurable text-fields, which can be completely defined at design time. Also, the statusbar provides standard fields like the Num-lock, Caps-lock and Scroll-lock toggles and a fully configurable dat/time field. The StatusBar control can also be used as a progress indicator. Besides that, the StatusBar is capable of automatically showing explanation messages in one of it's fields when the user is browsing through menu's.

Usage

To use the StatusBar, perform the following steps:

1. Add the toolbars custom control to your project. The StatusBar and the ButtonBar icons will appear in the Visual Basic tool-palette.

2. Add a StatusBar control to your form by double clicking on the icon in the ToolBox.

3. Define the fields for the StatusBar with the FieldProperties dialog or set the FieldWidth and FieldType properties for the number of fields you want to display in the Form_Load event procedure of your form and specify a Message for each field with FieldType 0.

4. Eventually add code to respond to a Click on a certain field in the StatusBar.

Distribution Note When you create and distribute applications that use the StatusBar control, you should install the file TOOLBARS.VBX in the customer's Microsoft Windows \ SYSTEM subdirectory.

Copyright Notice

The Toolbar Custom Controls, Version 1.52

Description

The Microsoft Visual Basic programming system for Windows comes with a large set of 3D controls. Unfortunately, a 3D status-bar and a 3D button-bar, as used in almost every MS-Windows application are lacking. Therefore, the Toolbar custom controls (TOOLBARS.VBX) were developed. This VBX contains two custom controls: the Status Bar Custom Control and the Button Bar custom control. These control allows you to create very versatile status- and buttonbars for all your applications.

File Name

TOOLBARS.VBX

Object Types

StatusBar, ButtonBar

Toolbox Icons


StatusBar control


ButtonBar control

Remarks

The StatusBar allows an application to display a status-bar at the bottom of a form. The statusbar has 20 fully configurable text-fields, which can be completely defined at design-time. Also, the statusbar provides standard fields like the Num-lock, Caps-lock and Scroll-lock toggles and a fully configurable date/time field. The StatusBar control can also be used as a progress indicator.

The ButtonBar allows an application to display a button-bar at the top of a form. The button-bar has 30 fully configurable buttons, which can be completely defined at design-time. The only thing you have to do is specify the bitmap for the up-position of each button. The ButtonBar control calculates the different bitmaps for the down and the two disabled states (up and down) of the button. Like all applications today, the ButtonBar supports tool-tips, a small window that pops up when the user rests the mouse-cursor on a button or a control placed on the ButtonBar that explains the function of the button (small hint). The ButtonBar calls these hints ButtonHints. It is also possible to connect the button-bar to the status-bar and specify (longer) messages for each button to be shown when the button is selected.

The ButtonBar also offers a PicClip like property 'ButtonPictures'. With this you can specify a bitmap that contains all the small bitmaps for the buttons on the ButtonBar. This avoids having to distribute all the small bitmaps that make up the ButtonBar since the bitmap is saved within your application. A little application (written in Visual Basic) that is distributed with the TOOLBARS.VBX allows easy creation and modification of such a large bitmap.

Distribution Note When you create and distribute applications that use one of the ToolBar controls, you should install the file TOOLBARS.VBX in the customer's Microsoft Windows \SYSTEM subdirectory.

See also

Click Event

See also

ButtonPicture Property
ButtonPictures Property
Picture Property Array
PictureDown Property Array
PictureDisabled Property Array
PictureDisabledDown Property Array

See also

ButtonPictures Property
ButtonColumns, ButtonRows Properties
Picture Property Array
PictureDown Property Array
PictureDisabled Property Array
PictureDisabledDown Property Array

See also

ButtonPicture Property
ButtonColumns, ButtonRows Properties
Picture Property Array
PictureDown Property Array
PictureDisabled Property Array
PictureDisabledDown Property Array

See also

ButtonHint Property Array
HintBackColor Property
HintDelay Property
HintOffsetX, HintOffsetY Properties
HintPosition Property
ShowHints Property
ShowDisabledHints Property

See also

ButtonHint Property Array
ControlHwnd, ControlHint, ControlMessage Properties
HintBackColor Property
HintDelay Property
HintPosition Property
ShowHints Property
ShowDisabledHints Property

See also

ButtonHint Property Array
ControlHwnd, ControlHint, ControlMessage Properties
HintBackColor Property
HintDelay Property
HintOffsetX, HintOffsetY Properties
ShowHints Property
ShowDisabledHints Property

See also

ButtonMessage Property Array
hWndStatusBar Property
StatusField Property
StatusBar Custom Control
ShowDisabledMessages Property

See also

HintBackColor Property
HintDelay Property
HintOffsetX, HintOffsetY Properties
HintPosition Property
ShowHints Property

See also

ButtonHint Property Array
ControlHwnd, ControlHint, ControlMessage Properties
HintBackColor Property
HintOffsetX, HintOffsetY Properties
HintPosition Property
ShowHints Property

See also

ButtonHint Property Array
ControlHwnd, ControlHint, ControlMessage Properties
FontBold Property
FontItalic Property
FontName Property
FontSize Property
FontStrikethru Property
FontUnderline Property
HintDelay Property
HintOffsetX, HintOffsetY Properties
HintPosition Property
ShowHints Property

See also

ButtonMessage Property Array
ButtonHint Property Array
ControlHwnd, ControlHint, ControlMessage Properties
FontBold Property
FontItalic Property
FontName Property
FontSize Property
FontStrikethru Property
FontUnderline Property
HintBackColor Property
HintDelay Property
HintOffsetX, HintOffsetY Properties
HintPosition Property
ShowDisabledHints Property

See also

ButtonHint Property Array
ControlHwnd, ControlHint, ControlMessage Properties
FontBold Property
FontItalic Property
FontName Property
FontStrikethru Property
FontUnderline Property
HintBackColor Property
HintDelay Property
HintOffsetX, HintOffsetY Properties
HintPosition Property
ShowHints Property
ShowDisabledHints Property

See also

ButtonHint Property Array
ControlHwnd, ControlHint, ControlMessage Properties
FontBold Property
FontItalic Property
FontSize Property
FontStrikethru Property
FontUnderline Property
HintBackColor Property
HintDelay Property
HintOffsetX, HintOffsetY Properties
HintPosition Property
ShowHints Property
ShowDisabledHints Property

See also

ButtonHint Property Array
ControlHwnd, ControlHint, ControlMessage Properties
FontName Property
FontSize Property
HintBackColor Property
HintDelay Property
HintOffsetX, HintOffsetY Properties
HintPosition Property
ShowHints Property
ShowDisabledHints Property

See also

ButtonEnabled Property Array
ButtonGroup Property Array
ButtonState Property Array
ButtonType Property Array
IgnoreInvisibleButtons Property

See also

ButtonEnabled Property Array
ButtonGroup Property Array
ButtonState Property Array
ButtonType Property Array
ButtonVisible Property Array

See also

ButtonGroup Property Array
ButtonState Property Array
ButtonType Property Array
ButtonVisible Property Array

See also

ButtonMessage Property Array
hWndStatusBar Property
ShowDisabledMessages Property
ShowStatusMessage Property
StatusBar Custom Control

See also

ButtonMessage Property Array
ShowStatusMessage Property
StatusField Property
StatusBar Custom Control

See also

ButtonPictures Property
ButtonPicture Property Array
ButtonColumns, ButtonRows Properties
ButtonEnabled Property Array
ButtonState Property Array
ButtonVisible Property Array
Picture Property Array
PictureDown Property Array
PictureDisabled Property Array

See also

ButtonPictures Property
ButtonPicture Property Array
ButtonColumns, ButtonRows Properties
ButtonEnabled Property Array
ButtonState Property Array
ButtonVisible Property Array
Picture Property Array
PictureDisabled Property Array
PictureDisabledDown Property Array

See also

ButtonPictures Property
ButtonPicture Property Array
ButtonColumns, ButtonRows Properties
ButtonEnabled Property Array
ButtonState Property Array
ButtonVisible Property Array
Picture Property Array
PictureDown Property Array
PictureDisabledDown Property Array

See also

ButtonPictures Property
ButtonPicture Property Array
ButtonColumns, ButtonRows Properties
ButtonEnabled Property Array
ButtonState Property Array
PictureDown Property Array
PictureDisabled Property Array
PictureDisabledDown Property Array

See also

ButtonGroup Property Array
ButtonState Property Array
ButtonType Property Array
ButtonVisible Property Array

See also

Picture Property Array

See also

hWndStatusBar Property
ShowDisabledMessages Property
ShowStatusMessage Property
StatusField Property
StatusBar Custom Control

See also

ButtonState Property Array
ButtonType Property Array
ButtonVisible Property Array
GroupAllowAllUp Property Array

See also

ButtonGroup Property Array
ButtonType Property Array
ButtonVisible Property Array
GroupAllowAllUp Property Array

See also

ButtonGroup Property Array
ButtonState Property Array
ButtonVisible Property Array
GroupAllowAllUp Property Array

See also

ButtonPictures Property
ButtonPicture Property Array
ButtonColumns, ButtonRows Properties
Picture Property Array
PictureDown Property Array
PictureDisabled Property Array
PictureDisabledDown Property Array

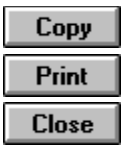
See also

ButtonEnabled Property Array
ButtonMessage Property Array
ShowStatusMessage Property

See also

ButtonEnabled Property Array
ButtonHint Property Array
ShowHints Property

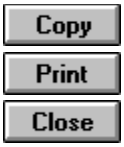
Button/ControlHints Example, ButtonBar Control




```
Sub Form_Load ()
Dim I As Integer
Const BUTTON_NORMAL = 0
Const BUTTON_2STATE = 1

Const BELOW_RIGHT = 4

' Fill Combo1 with all available fonts
For I = 0 To Screen.FontCount - 1
    Combo1.AddItem Screen.Fonts(I)
Next I

' Select the first font in the combo-box
Combo1.ListIndex = 0

' Align the button to the top of the window
' and set the height
ButtonBar1.Align = 1
ButtonBar1.Height = 400

' Set the fontproperties for the Button-Hint popup
ButtonBar1.FontName = "Times New Roman"
ButtonBar1.FontSize = 12
ButtonBar1.FontBold = False

' Make sure the button-hints appear (after half a second) and set the
' backcolor for the hint-window to light-blue
ButtonBar1.ShowHints = True
ButtonBar1.HintDelay = 500
ButtonBar1.HintBackColor = QBColor(9)

' Set the Hint-position and offsets
ButtonBar1.HintPosition = BELOW_RIGHT
ButtonBar1.HintOffsetX = -4
ButtonBar1.HintOffsetY = 2

' Set the button-pictures.
ButtonBar1.Picture(0)=LoadPicture("c:\vb\bitmaps\toolbar3\tbl-up.bmp")
ButtonBar1.Picture(1)=LoadPicture("c:\vb\bitmaps\toolbar3\tbc-up.bmp")
ButtonBar1.Picture(2)=LoadPicture("c:\vb\bitmaps\toolbar3\tbr-up.bmp")
ButtonBar1.Picture(3)=LoadPicture("c:\vb\bitmaps\toolbar3\tbd-up.bmp")

' Set the button-hints for each button
ButtonBar1.ButtonHint(0)= "left-tab"
ButtonBar1.ButtonHint(1)= "center-tab"
ButtonBar1.ButtonHint(2)= "right-tab"
ButtonBar1.ButtonHint(3)= "decimal-tab"

' Set the ControlHint for the fonts combo-box
```

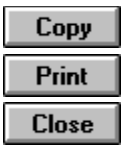
```
' Combo1 must be defined on the ButtonBar control
ButtonBar1.ControlHwnd = Combo1.hWnd
ButtonBar1.ControlHint = "select font"

' Make the first four buttons a member of group 0
ButtonBar1.ButtonGroup(0) = 0
ButtonBar1.ButtonGroup(1) = 0
ButtonBar1.ButtonGroup(2) = 0
ButtonBar1.ButtonGroup(3) = 0

' Specify that in group 0, all buttons may be raised
ButtonBar1.GroupAllowAllUp(0) = True

' Mark buttons 0-3 as 2-state buttons
For I = 0 To 3
    ButtonBar1.ButtonType(I) = BUTTON_2STATE
Next I

End Sub
```

ButtonPictures example, ButtonBar Control




```
' to use this example, create a form with a buttonbar control that
' contains a bitmap that contains the bitmaps for the different
' states of each button on the ButtonBar.
' You can use the sample bitmap BUTTONS.BMP that comes with the
' TOOLBARS.VBX

Sub Form_Load ()
Dim I As Integer
Dim CelStart As Integer

 ButtonBar1.ButtonPictures = LoadPicture("BUTTONS.BMP")
 ButtonBar1.ButtonRows = 30
 ButtonBar1.ButtonColumns = 4

 For I = 0 To 29
  CelStart = I * 4
  ButtonBar1.Picture(I) = ButtonBar1.ButtonPicture(CelStart)
  ButtonBar1.PictureDisabled(I) = ButtonBar1. ButtonPicture(CelStart + 1)
  ButtonBar1.PictureDown(I) = ButtonBar1. ButtonPicture(CelStart + 2)
  ButtonBar1.PictureDisabledDown(I) = ButtonBar1. ButtonPicture(CelStart +
3)
 Next I
End Sub
```

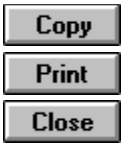
Usage with the StatusBar Control, ButtonBar Control




```
Sub Form_Load ()
Dim I As Integer

Const BUTTON_2STATE = 1

' Constants for ShowStatusMessage property
Const NEVER = 0
Const ONSELECT = 1
Const MOUSEOVER = 2
Const RIGHTMOUSE = 3

WindowState = 2
ButtonBar1.Align = 1
ButtonBar1.Height = 400
StatusBar1.Align = 2
StatusBar1.Height = 400

' Specify the pictures for the first four buttons
ButtonBar1.Picture(0)=LoadPicture("c:\vb\bitmaps\toolbar3\tbl-up.bmp")
ButtonBar1.Picture(1)=LoadPicture("c:\vb\bitmaps\toolbar3\tbc-up.bmp")
ButtonBar1.Picture(2)=LoadPicture("c:\vb\bitmaps\toolbar3\tbr-up.bmp")
ButtonBar1.Picture(3)=LoadPicture("c:\vb\bitmaps\toolbar3\tbd-up.bmp")

' Make all buttons members of group 0
ButtonBar1.ButtonGroup(0) = 0
ButtonBar1.ButtonGroup(1) = 0
ButtonBar1.ButtonGroup(2) = 0
ButtonBar1.ButtonGroup(3) = 0
' Allow all buttons in the group to be up.
ButtonBar1.GroupAllowAllUp(0) = True

' Set all buttons to type 1 (2-state button)
For I = 0 To 3
    ButtonBar1.ButtonType(I) = BUTTON_2STATE
Next I

' Specify messages to display in the statusbar
ButtonBar1.ButtonMessage(0) = "Set a left align tab"
ButtonBar1.ButtonMessage(1) = "Set a centered tab"
ButtonBar1.ButtonMessage(2) = "Set a right align tab"
ButtonBar1.ButtonMessage(3) = "Set a decimal tab"

' Specify the statusbar and the field to display the messages
ButtonBar1.hWndStatusBar = StatusBar1.hWnd
ButtonBar1.StatusField = 0

' Make sure the messages appear when the mouse is over a button
ButtonBar1.ShowStatusMessage = MOUSEOVER
```

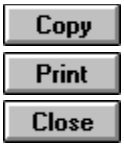
```
' Set a few properties of the statusbar control
StatusBar1.FieldWidth(0) = 100
StatusBar1.Message(0) = "Static message for field 0"

End Sub
```

ButtonTag Example, ButtonBar Control




```
Sub Form_Load ()
Const BUTTON_NORMAL = 0
Const BUTTON_2STATE = 1

 WindowState = 2
 ButtonBar1.Align = 1
 ButtonBar1.Height = 400

 ButtonBar1.Picture(0)=LoadPicture("c:\vb\bitmaps\toolbar\lft-up.bmp")
 ButtonBar1.ButtonTag(0) = "Left Align"
 ButtonBar1.Picture(1)=LoadPicture("c:\vb\bitmaps\toolbar\rt-up.bmp")
 ButtonBar1.ButtonTag(1) = "Right Align"
 ButtonBar1.Picture(2)=LoadPicture("c:\vb\bitmaps\toolbar\cnt-up.bmp")
 ButtonBar1.ButtonTag(2) = "Center"
 ButtonBar1.PictureDisabled(0)=LoadPicture("c:\vb\bitmaps\toolbar\lft-
dis.bmp")
 ButtonBar1.PictureDisabled(1)=LoadPicture("c:\vb\bitmaps\toolbar\rt-
dis.bmp")
 ButtonBar1.PictureDisabled(2)=LoadPicture("c:\vb\bitmaps\toolbar\cnt-
dis.bmp")
 ButtonBar1.ButtonType(0) = BUTTON_2STATE
 ButtonBar1.ButtonType(1) = BUTTON_2STATE
 ButtonBar1.ButtonType(2) = BUTTON_2STATE
 ButtonBar1.ButtonGroup(0) = 0
 ButtonBar1.ButtonGroup(1) = 0
 ButtonBar1.ButtonGroup(2) = 0

 ButtonBar1.ButtonState(0) = True
 ButtonBar1.GroupAllowAllUp(0) = False
End Sub

Sub ButtonBar1_Click(Button As Integer, Group As Integer, State As Integer)
    Select Case ButtonBar1.ButtonTag(Button)
          Case "Left Align": MsgBox "Text will be left aligned"
          Case "Right Align": MsgBox "Text will be right aligned"
          Case "Center": MsgBox "Text will be centered"
    End Select
End Sub
```

Click Example, ButtonBar Control




```
Sub Form_Load ()
Const BUTTON_NORMAL = 0
Const BUTTON_2STATE = 1

 WindowState = 2
 ButtonBar1.Align = 1
 ButtonBar1.Height = 400

 ButtonBar1.Picture(0)=LoadPicture("c:\vb\bitmaps\toolbar\lft-up.bmp")
 ButtonBar1.Picture(1)=LoadPicture("c:\vb\bitmaps\toolbar\rt-up.bmp")
 ButtonBar1.Picture(2)=LoadPicture("c:\vb\bitmaps\toolbar\cnt-up.bmp")
 ButtonBar1.PictureDisabled(0)=LoadPicture("c:\vb\bitmaps\toolbar\lft-
dis.bmp")
 ButtonBar1.PictureDisabled(1)=LoadPicture("c:\vb\bitmaps\toolbar\rt-
dis.bmp")
 ButtonBar1.PictureDisabled(2)=LoadPicture("c:\vb\bitmaps\toolbar\cnt-
dis.bmp")
 ButtonBar1.ButtonType(0) = BUTTON_2STATE
 ButtonBar1.ButtonType(1) = BUTTON_2STATE
 ButtonBar1.ButtonType(2) = BUTTON_2STATE
 ButtonBar1.ButtonGroup(0) = 0
 ButtonBar1.ButtonGroup(1) = 0
 ButtonBar1.ButtonGroup(2) = 0

 ButtonBar1.ButtonEnabled(2)=False

 ButtonBar1.ButtonState(0) = True
 ButtonBar1.GroupAllowAllUp(0) = False
End Sub

Sub ButtonBar1_Click(Button As Integer, Group As Integer, State As Integer)
    Label1.Alignment = Button
    Select Case Button
          Case 0: Label1.Caption = "Text is now left-aligned"
          Case 1: Label1.Caption = "Text is now right-aligned"
          Case 2: 'Should never get here since the centered button is
disabled
                  Label1.Caption = "Text is now centered"
    End Select
End Sub
```

ButtonBar Initialize Example, ButtonBar Control

Copy
Print
Close

```
Sub Form_Load ()
Dim I As Integer
Const BUTTON_NORMAL = 0
Const BUTTON_2STATE = 1

Const RAISED = 0
Const DEPRESSED = -1

' Align the button to the top of the window
' and set the height
ButtonBar1.Align = 1
ButtonBar1.Height = 400

' Set the first four button-pictures.
ButtonBar1.Picture(0)=LoadPicture("c:\vb\bitmaps\toolbar3\tbl-up.bmp")
ButtonBar1.Picture(1)=LoadPicture("c:\vb\bitmaps\toolbar3\tbc-up.bmp")
ButtonBar1.Picture(2)=LoadPicture("c:\vb\bitmaps\toolbar3\tbr-up.bmp")
ButtonBar1.Picture(3)=LoadPicture("c:\vb\bitmaps\toolbar3\tbd-up.bmp")
' Specify five pixels room after button #3
ButtonBar1.SpaceAfter(3) = 5
' Make the first four buttons a member of group 0
ButtonBar1.ButtonGroup(0) = 0
ButtonBar1.ButtonGroup(1) = 0
ButtonBar1.ButtonGroup(2) = 0
ButtonBar1.ButtonGroup(3) = 0
' Specify that in group 0, all buttons may be raised
ButtonBar1.GroupAllowAllUp(0) = True

' Create four extra buttons
ButtonBar1.Picture(4)=LoadPicture("c:\vb\bitmaps\toolbar3\lft-up.bmp")
ButtonBar1.Picture(5)=LoadPicture("c:\vb\bitmaps\toolbar3\cnt-up.bmp")
ButtonBar1.Picture(6)=LoadPicture("c:\vb\bitmaps\toolbar3\rt-up.bmp")
ButtonBar1.Picture(7)=LoadPicture("c:\vb\bitmaps\toolbar3\jst-up.bmp")
' Leave 5 pixels space after button #7
ButtonBar1.SpaceAfter(7) = 5
' Make the second four buttons a member of group 1
ButtonBar1.ButtonGroup(4) = 1
ButtonBar1.ButtonGroup(5) = 1
ButtonBar1.ButtonGroup(6) = 1
ButtonBar1.ButtonGroup(7) = 1
' Specify that at least one button of the group must be depressed
ButtonBar1.GroupAllowAllUp(1) = False

' Make three extra buttons
ButtonBar1.Picture(8)=LoadPicture("c:\vb\bitmaps\toolbar3\bld-up.bmp")
ButtonBar1.Picture(9)=LoadPicture("c:\vb\bitmaps\toolbar3\itl-up.bmp")
ButtonBar1.Picture(10)=LoadPicture("c:\vb\bitmaps\toolbar3\ulin-up.bmp")

' Leave 5 pixels room after button 10
```

```
ButtonBar1.SpaceAfter(10) = 5

' Mark buttons 0-10 as 2-state buttons
For I = 0 To 10
    ButtonBar1.ButtonType(I) = BUTTON_2STATE
Next I

' Press button 4 down
ButtonBar1.ButtonState(4) = True

' Create the last button
ButtonBar1.Picture(11)=LoadPicture("c:\vb\bitmaps\toolbar3\hlp-up.bmp")
End Sub
```

ControlHwnd, ControlHint, ControlMessage Properties, ButtonBar Control

see also example

Description

Sets or retrieves the text to show in the Hint popup-window and the text to show in a field of the StatusBar when the mouse is over a Control in the ButtonBar.

Usage

[*form!*]*ButtonBar1*.**ControlHwnd** [= *setting%*]

[*form!*]*ButtonBar1*.**ControlHint** [= *setting$*]

[*form!*]*ButtonBar1*.**ControlMessage** [= *setting$*]

Remarks

The ButtonBar can automatically display an explanation-text for each button or control you have on it. The exaplanation text is shown after a certain period (see the HintDelay property) when the mouse is over a button. The Hint is shown in a small window just below or above the button.

Since there is no control array of controls you place on the ButtonBar, you can't specify a array-entry for a ControlHint (like in the ButtonHint property array). Therefore, the ControlHwnd and ControlHint properties work together to let you specify the Hints for every control you've placed on the ButtonBar.

First, you specify for which control on the ButtonBar you want to set a Hint-message, by setting the ControlHwnd property (e.g. ButtonBar1.ControlHwnd = Combo1.hWnd). This makes the control 'active'. After that, you can set a Hint-message for the 'active' control by setting the ControlHint property.

The Hints are only shown is the ButtonBar's ShowHints property is set to TRUE.

Furthermore, you can specify a message to be shown in a field of the StatusBar if the ButtonBar is connected to one. Here also, the ControlHwnd and ControlMessage properties work together to let you specify the messages for the StatusBar. See the ButtonMessage property array for details.

Data Type

Integer, String

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control that use ButtonHints. To use this example create a form with one ButtonBar control. Place a combo-box on the ButtonBar and paste the code into the Declarations section of your form.

ButtonHint Property Array, ButtonBar Control

see also example

Description

Sets or retrieves the text to show in the ButtonHint popup-window when the mouse is over a button in the ButtonBar.

Usage

[*form!*]*ButtonBar1*.**ButtonHint(I)** [= *setting$*]

Remarks

The ButtonBar can automatically display an explanation-text for each button or control you have on it. The exaplanation text is shown after a certain period (see the HintDelay property) when the mouse is over a button. The ButtonHint is shown in a small window just below or above the button. To specify a message for a control placed on the ButtonBar use the ControlHwnd and ControlHint properties.

Don't confuse the ButtonHint with the ButtonMessage property, because they're not the same. The ButtonHint is usually a small-text displayed in a small popup-window just below the button, whereas the ButtonMessage can be very large and is displayed in a field on the StatusBar control.

The ButtonHints are only shown if the ButtonBar's ShowHints property is set to TRUE.

Data Type

String

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control that use ButtonHints. To use this example create a form with one ButtonBar control. Place a combo-box on the ButtonBar and paste the code into the Declarations section of your form.

HintOffsetX, HintOffsetY Properties, ButtonBar Control

see also example

Description

Sets or retrieves the relative offset from the Button or Control where the ButtonBar will position the ButtonHint popup-window depending on the setting of the HintPosition property.

Usage

[*form!*]*ButtonBar1*.**HintOffsetX** [= *setting%*]

[*form!*]*ButtonBar1*.**HintOffsetY** [= *setting%*]

Remarks

You can use the HintOffsetX and HintOffsetY properties to further specify the position where the ButtonBar will display the Button/ControlHint for the button or control the mouse is currently over. You can control the distance from the button or control by setting the HintOffsetX and HintOffsetY properties. The HintOffsetX modifies the horizontal position of the Hint-popup window, the HintOffsetY property modifies the vertical position of the Hint-popup window.

The HintOffsetY specifies the distance the Hint-window will have from the button or control it is currently displayed for. So, a bigger value will cause the Hint-popup window to be further apart from the button or control. A negative value will cause the Hint-popup and the control to overlap.

These properties have a default value of 4.

Data Type

Integer

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control that use ButtonHints. To use this example create a form with one ButtonBar control. Place a combo-box on the ButtonBar and paste the code into the Declarations section of your form.

HintPosition Property, ButtonBar Control

see also example

Description

Sets or retrieves the position where the ButtonBar will position the ButtonHint popup-window.

Usage

[*form!*]*ButtonBar1*.**HintPosition** [= *setting%*]

Remarks

You can use the HintPosition property to specify the position where the ButtonBar will display the Button/ControlHint for the button or control the mouse is currently over. You can control the distance from the button or control by setting the HintOffsetX and HintOffsetY properties.

The HintPosition property takes one of the following values to specify where the ButtonBar shows the defined Button/ControlHints:

Value	Short name	Description
0	ABOVE_LEFT	Displays the Hint popup window aligned to the left and above the button or control the mouse is over.
1	ABOVE_RIGHT	Displays the Hint popup window aligned to the right and above the button or control the mouse is over.
2	ABOVE_CENTER	Displays the Hint popup window centered and above the button or control the mouse is over.
3	BELOW_LEFT	(Default) Displays the Hint popup window aligned to the left and below the button or control the mouse is over.
4	BELOW_RIGHT	Displays the Hint popup window aligned to the right and below the button or control the mouse is over.
5	BELOW_CENTER	Displays the Hint popup window centered and below the button or control the mouse is over.

Data Type

Integer (Enumerated)

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control that use ButtonHints. To use this example create a form with one ButtonBar control. Place a combo-box on the ButtonBar and paste the code into the Declarations section of your form.

HintDelay Property, ButtonBar Control

see also example

Description

Sets or retrieves the time (in milliseconds) the ButtonBar will wait before displaying the ButtonHint popup-window.

Usage

[*form!*]*ButtonBar1*.**HintDelay** [= *setting&*]

Remarks

You can use the HintDelay property to specify the number of milliseconds the ButtonBar will wait before displaying the ButtonHint for the button or control the mouse currently is over. Once the specified delay has past and the mouse is still over the same button or control, the Hint popup-window is displayed right below or above that button. The Hint popup-window remains visible as long as the mouse-pointer is over one of the buttons or controls on the ButtonBar. If you move the mouse to a position where no button or control is present, the Hint popup will disappear.

Data Type

Long

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control that use ButtonHints. To use this example create a form with one ButtonBar control. Place a combo-box on the ButtonBar and paste the code into the Declarations section of your form.

HintBackColor Property, ButtonBar Control

see also example

Description

Sets or retrieves the backcolor of the ButtonHint popup-window.

Usage

[*form!*]*ButtonBar1*.**HintBackColor** [= *color&*]

Remarks

Visual Basic uses the Microsoft Windows environment RGB scheme for colors. The HintBackColor property has the following ranges of settings:

Range of settings	Description
Normal RGB colors	Colors by using the RGB or QBColor functions in code.
System default colors	Colors specified with system color constants from CONSTANT.TXT, a Visual Basic file that specifies system defaults. The Windows environment substitutes the user's choices as specified in the user's Control Panel settings.

For the ButtonBar control the default setting for the Hint popup-window is RGB(255,255,128), which is light-yellow.

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower three bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte is not 0, Visual Basic uses the system colors, as defined in the user's Control Panel and enumerated in CONSTANT.TXT.

Data Type

Long

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control that use ButtonHints. To use this example create a form with one ButtonBar control. Place a combo-box on the ButtonBar and paste the code into the Declarations section of your form.

ShowHints Property, ButtonBar Control

see also example

Description

Specifies whether or not to show the Button/ControlHint-popup windowif the mouse-cursor is over a button or a control on the ButtonBar.

Usage

[*form!*]*ButtonBar1*.**ShowHints** [= *setting%*]

Remarks

The ButtonBar can automatically display an explanation-text for each button and control you have on it. The exaplanation text is shown after a certain period (see the HintDelay property) when the mouse is over a button or control defined on the buttonbar. The ButtonHint or ControlHint is shown in a small window just below or above the button or control, depending on the setting of the HintPosition and HintOffsetX and HintOffsetY properties.

Don't confuse the ButtonHint with the ButtonMessage property, because they're not the same. The ButtonHint is usually a small-text displayed in a small popup-window just below or above the button, whereas the ButtonMessage can be very large and is displayed in a field on the StatusBar control.

The ShowHints property takes one of the following values to specify the if the ButtonBar shows the defined ButtonHints.

Value	Short name	Description
0	FALSE	(Default) Does not display Button/ControlHints.
-1	TRUE	Displays the Button/ControlHint window after a small period of time (usually half a second) for the button or control the cursor is on.

Data Type

Integer (Boolean)

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control that use ButtonHints. To use this example create a form with one ButtonBar. Place a combo-box on the ButtonBar control and paste the code into the Declarations section of your form.

ShowDisabledMessages Property, ButtonBar Control

see also

Description

Specifies whether or not to show the Button or ControlMessage for disabled buttons or controls.

Usage

[*form!*]*ButtonBar1*.**ShowDisabledMessages** [= *setting%*]

Remarks

The ButtonBar can automatically display an explanation-text for each button and control you have on it in a textfield of a StatusBar control.. Whether these messages are shown depends on the settings of the ShowStatusMessage property and the ShowDisabledMessages property.

The ShowDisabledMessages property takes one of the following values to specify the if the ButtonBar shows the defined Button- or ControlHints for disabled buttons or controls.

Value	Short name	Description
0	FALSE	(Default) Does not display Button/ControlMessages if the button or control is disabled. If the ShowStatusMessage property is set to True, Button/ControlMessages for all other buttons and controls are displayed.
-1	TRUE	Displays the Button/ControlMessage for all buttons or controls on the ButtonBar even if they are disabled, if the ShowStatusMessage property is also set to True.

Data Type

Integer (Boolean)

ShowDisabledHints Property, ButtonBar Control

see also

Description

Specifies whether or not to show the Button/ControlHint-popup window for disabled buttons or controls.

Usage

[*form!*]*ButtonBar1*.**ShowDisabledHints** [= *setting%*]

Remarks

The ButtonBar can automatically display an explanation-text for each button and control you have on it. Whether these messages are shown depends on the settings of the ShowHints property and the ShowDisabledHints property.

The ShowDisabledHints property takes one of the following values to specify the if the ButtonBar shows the defined Button- or ControlHints for disabled buttons or controls.

Value	Short name	Description
0	FALSE	(Default) Does not display Button/ControlHints if the button or control is disabled. If the ShowHints property is set to True, Button/ControlHints for all other buttons and controls are displayed.
-1	TRUE	Displays the Button/ControlHint for all buttons or controls on the ButtonBar even if they are disabled, if the ShowHints property is also set to True.

Data Type

Integer (Boolean)

Click Event, ButtonBar Control

see also example

Description

Occurs when the user presses and then releases the left-mouse button over a button of the ButtonBar control. You can not trigger the Click event for the ButtonBar control in code.

Syntax

Sub *ButtonBar1*_**Click** (*Index* **As Integer**, *Button* **As Integer,** *Group* **As Integer,** *State* **As Integer**)

Remarks

The argument *Index* uniquely identifies a control if it is in a control array. The *Button* argument specifies the number of the button that was clicked on. You can use this number to take some action whenever the user clicks on a button. The *Group* argument specifies the group the clicked button is a member of. If the *Group* argument is -1, the button is not a member of a group. The State argument specifies the state of the button. This argument is always True (-1) if the ButtonType of the button is 0 (BUTTON_NORMAL). If the ButtonType of the clicked button is 1 (BUTTON_2STATE) then the *State* argument specifies whether the button is up (State = False) or down (State = True).

The example shows you a possible way to react to the Click event. To use this example create a form with one ButtonBar control and a Label control. Then, paste the code into the Declarations section of your form.

FontSize Property, ButtonBar Control

see also example

Description

Determines the size of the font to be used for the Hints-popup.

Usage

[*form!*]*ButtonBar1*.**FontSize** [= *setting*]

Remarks

Use this font property to specify the size of the text displayed in the Hint floating window.

Note

Fonts available in Visual Basic vary according to your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which actual fonts exist.In general, you should change the FontName property before you set size and style attributes with the FontSize, FontBold, FontItalic, FontStrikethru, and FontUnderline properties.

However, when you set TrueType fonts to smaller than 8 points, you should set the point size with the FontSize property, then set the FontName property, and then set the size again with the FontSize property. The Windows environment uses a different font for TrueType fonts that are smaller than 8 points.

Data Type

Single

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control that use ButtonHints with a different FontSize than normal. To use this example create a form with one ButtonBar control. Place a combo-box on the ButtonBar and paste the code into the Declarations section of your form.

FontBold, FontItalic, FontStrikethru and FontUnderline Properties, ButtonBar Control

see also example

Description

Determine font styles for the Button/ControlHints in the following formats: **FontBold**, *FontItalic*, ~~FontStrikethru~~, and FontUnderline.

Usage

[*form!*]*ButtonBar1*.**FontBold** [= *setting%*]

[*form!*]*ButtonBar1*.**FontItalic** [= *setting%*]

[*form!*]*ButtonBar1*.**FontStrikethru** [= *setting%*]

[*form!*]*ButtonBar1*.**FontUnderline** [= *setting%*]

Remarks

The settings for the FontBold, FontItalic, FontStrikethru, and FontUnderline properties are:

Setting	Description
True	Turns on the formatting in that style.
False	(default) Turns off the formatting in that style.

Use these font properties to format the look of the text displayed in the Hint floating window.

Note

Fonts available in Visual Basic vary according to your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which actual fonts exist.In general, you should change the FontName property before you set size and style attributes with the FontSize, FontBold, FontItalic, FontStrikethru, and FontUnderline properties.

However, when you set TrueType fonts to smaller than 8 points, you should set the point size with the FontSize property, then set the FontName property, and then set the size again with the FontSize property. The Windows environment uses a different font for TrueType fonts that are smaller than 8 points.

Data Type

Integer (Boolean)

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control that use ButtonHints with a different text-layout than normal. To use this example create a form with one ButtonBar control. Place a combo-box on the ButtonBar and paste the code into the Declarations section of your form.

FontName Property, ButtonBar Control

see also example

Description

Determines the font used to display the Hints.

Usage

[*form!*]*ButtonBar1*.**FontName** [= *setting$*]

Remarks

The default for this property is determined by the system. Fonts available with Visual Basic vary according to your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which fonts exist.

In general, you should change FontName before setting size and style attributes with the FontSize, FontBold, FontItalic, FontStrikethru and FontUnderline properties.

On systems running Windows 3.0, the fonts "Helv" or "Tms Rmn" are called "MS Sans Serif" and "MS Serif", respectively. In code, if you set FontName to "Helv," then test whether the FontName is set to "Helv," the result will be False, since it will be changed internally to "MS Sans Serif."

Data Type

String (Boolean)

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control that use ButtonHints with a different FontName than normal. To use this example create a form with one ButtonBar control. Place a combo-box on the ButtonBar and paste the code into the Declarations section of your form.

OutlineChildren Property, ButtonBar Control

Description

Apllies a 3D look to controls placed on the ButtonBar.

Usage

[*form!*]*ButtonBar1*.**OutlineChildren** [= *setting%*]

Remarks

The ButtonBar can automatically give all the controls you place on it a 3D look by drawing a raised or inset outline. The OutlineChildren property takes one of the following values to specify the way the ButtonBar applies the 3D effect:

Value	Short name	Description
0	NONE	(Default) Does not draw an outline around children.
1	RAISED	Displays a raised border around each control placed on the ButtonBar.
2	INSET	Displays an inset border around each controls placed on the ButtonBar.

The 3D effect is useful when you want to put more than just buttons on the ButtonBar (like a font-selection combo-box). These control can appear raised or inset just by setting the OutlineChildren property.

The OutlineChildren property only has effect on controls that have a window-handle. Therefore, graphical controls (like labels) do not receive an outline.

Warning

Since the ButtonBar has to keep track of all children placed on it and their position, the maximum number of windowed child controls (that is; controls with a window-handle) on the ButtonBar is limited to 30.

Data Type

Integer (Enumerated)

StatusField Property, ButtonBar Control

see also example

Description

Specifies the text-field of the status-bar that is to display the messages specified for the buttons.

Usage

[*form!*]*ButtonBar1*.**StatusField** [= *setting%*]

Remarks

The StatusField property sets or retrieves the text-field of the status-bar that is to display the messages you specify in the ButtonMessage property array. If the text-field has a FieldWidth of zero, the messages are not displayed.

The messages are displayed in the statusbar specified by the hWndStatusBar property.

Data Type

Integer

The example shows a Form_Load that initializes some buttons with messages on a ButtonBar control. To use this example create a form with one ButtonBar and one StatusBar control and paste the code into the Declarations section of your form.

hWndStatusBar Property, ButtonBar Control

see also example

Description

Specifies the window handle of the status-bar that is to display the messages specified for the buttons.

Usage

[*form!*]*ButtonBar1*.**hWndStatusBar** [= [*form!*].*StatusBar1*.**hWnd**]

Remarks

The hWndStatusBar property sets or retrieves the window handle for the status-bar that is to display the messages you specify in the ButtonMessage property array. The window-handle is checked to see if it is really the window handle of a statusbar control. If it isn't, a runtime error is generated.

The messages are displayed in the statusbar in the text-field specified by the StatusField property. This property defaults to 0.

Data Type

Integer

The example shows a Form_Load that initializes some buttons with messages on a ButtonBar control. To use this example create a form with one ButtonBar and one StatusBar control and paste the code into the Declarations section of your form.

PictureDisabledDown Property Array, ButtonBar Control

see also example ButtonPictures example

Description

Defines the picture to use for showing the down state of a disabled button on the ButtonBar.

Usage

[*form!*]*ButtonBar1*.**PictureDisabledDown(I)** [= *picture*]

The Picture property settings are:

Setting	Description
(none)	(Default) No picture.
(bitmap)	You can set this property using the LoadPicture function on a bitmap.

Remarks

If you don't like the default look of the down-state of a disabled button, you can set the PictureDisabledDown property to another bitmap that has a different disabled down-state look. Like the Picture property array, the picture is assumed to be a bitmap that is already formatted as a button.

The ButtonBar control makes some assumptions about the format of the picture you specify for the PictureDown property in order to be able to create the different pictures needed for the other states of the button. All pictures in the TOOLBAR3 directory that come with Visual Basic have a correct layout. You can use or modify these pictures as needed.

Data Type

Integer

The example shows a Form_Load that initializes some buttons on a ButtonBar control. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

Hint Since you have to set a picture for each button, you can use the ButtonPictures property of the ButtonBar to define a bitmap that contains all the button-pictures. You can use the sample bitmap BUTTONS.BMP that comes with the TOOLBARS.VBX for this. This allows you to load all the bitmaps needed for the ButtonBar in a single for-loop. An example clarifies the use.

PictureDown Property Array, ButtonBar Control

see also example ButtonPictures example

Description

Defines the picture to use for showing the down state of a button on the ButtonBar.

Usage

[*form!*]*ButtonBar1*.**PictureDown(I)** [= *picture*]

The Picture property settings are:

Setting	Description
(none)	(Default) No picture.
(bitmap)	You can set this property using the LoadPicture function on a bitmap.

Remarks

If you don't like the default look of the down-state of a button, you can set the PictureDown property to another bitmap that has a different down-state look. Like the Picture property array, the picture is assumed to be a bitmap that is already formatted as a button.

The ButtonBar control makes some assumptions about the format of the picture you specify for the PictureDown property in order to be able to create the different pictures needed for the other states of the button. All pictures in the TOOLBAR3 directory that come with Visual Basic have a correct layout. You can use or modify these pictures as needed.

Data Type

Integer

The example shows a Form_Load that initializes some buttons on a ButtonBar control. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

Hint Since you have to set a picture for each button, you can use the ButtonPictures property of the ButtonBar to define a bitmap that contains all the button-pictures. You can use the sample bitmap BUTTONS.BMP that comes with the TOOLBARS.VBX for this. This allows you to load all the bitmaps needed for the ButtonBar in a single for-loop. An example clarifies the use.

PictureDisabled Property Array, ButtonBar Control

see also example ButtonPictures example

Description

Defines the picture to use for showing a disabled button on the ButtonBar.

Usage

[*form!*]*ButtonBar1*.**PictureDisabled(I)** [= *picture*]

The Picture property settings are:

Setting	Description
(none)	(Default) No picture.
(bitmap)	You can set this property using the LoadPicture function on a bitmap.

Remarks

If you don't like the default look of the disabled pictures (all black pixels transformed to gray), you can set the PictureDisabled property to another bitmap that has a different disabled look. Like the Picture property array, the picture is assumed to be a bitmap that is already formatted as a button.

When the PictureDisabled property of the button is set, and no PictureDisabledDown picture is present for button, the ButtonBar uses this picture to calculate the pictures for the disabled up and down states of the button.

The ButtonBar control makes some assumptions about the format of the picture you specify for the PictureDisabled property in order to be able to create the different pictures needed for the other states of the button. All pictures in the TOOLBAR3 directory that come with Visual Basic have a correct layout. You can use or modify these pictures as needed.

Data Type

Integer

The example shows a Form_Load that initializes some buttons on a ButtonBar control. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

Hint Since you have to set a picture for each button, you can use the ButtonPictures property of the ButtonBar to define a bitmap that contains all the button-pictures. You can use the sample bitmap BUTTONS.BMP that comes with the TOOLBARS.VBX for this. This allows you to load all the bitmaps needed for the ButtonBar in a single for-loop. An example clarifies the use.

Picture Property Array, ButtonBar Control

see also | example | ButtonPictures example

Description

Defines the picture to use for showing a button on the ButtonBar.

Usage

[*form!*]*ButtonBar1*.**Picture(I)** [= *picture*]

The Picture property settings are:

Setting	Description
(none)	(Default) No picture.
(bitmap)	You can set this property using the LoadPicture function on a bitmap.

Remarks

The Picture property array is the most important property for the ButtonBar. It specifies the picture to be used for a button on the ButtonBar. The picture is assumed to be a bitmap that is already formatted as a button.

If there are no pictures specified for the PictureDown, PictureDisabled and PictureDisabledDown property arrays for the button, the ButtonBar uses the picture in the Picture property array to calculate the pictures for the down and disabled states of the button.

The ButtonBar control makes some assumptions about the format of the picture you specify for the Picture property in order to be able to create the different pictures needed for the other states of the button. All pictures in the TOOLBAR3 directory that come with Visual Basic have a correct layout. You can use or modify these pictures as needed.

The following table specifies the operations that the ButtonBar will perform on a bitmap for the different ButtonStates and ButtonEnabled properties of the button if there are no pictures specified for the down, disabled and disabled-down states:

ButtonState	ButtonEnabled	Description
UP (0)	True	Value from the Picture property unchanged.
UP	False	Value from the Picture property with all black pixels transformed to dark-gray.
DOWN (-1)		True Value from the Picture property shifted one pixel to the right and below. The dark-gray bevel is removed and the white bevel is replaced with a dark gray bevel.
DOWN	False	Value from the Picture property modified as with the ButtonEnabled property set to True, but with all black pixels transformed to dark-gray.

Data Type

Integer

The example shows a Form_Load that initializes some buttons on a ButtonBar control. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

Hint Since you have to set a picture for each button, you can use the ButtonPictures property of the ButtonBar to define a bitmap that contains all the button-pictures. You can use the sample bitmap BUTTONS.BMP that comes with the TOOLBARS.VBX for this. This allows you to load all the bitmaps needed for the ButtonBar in a single for-loop. An example clarifies the use.

GroupAllowAllUp Property Array, ButtonBar Control

see also example

Description

Sets or retrieves the ability of buttons in a group to be all in the up-state.

Usage

[*form!*]*ButtonBar1*.**GroupAllowAllUp(I)** [= *setting%*]

Remarks

The GroupAllowAllUp property settings are:

Setting	Description
True	(Default) Allows all buttons in the group to be raised.
False	At least one button in the group must be depressed.

The GroupAllowAllUp property specifies if the user can deselect all buttons in a group, or that at least one button must be in the down position. Normally, when buttons are selected in a group, there is always one button in the down state. Clicking this button will leave it in the down position. When you specify True for the GroupAllowAllUp property, the user can deselect all buttons, by clicking the button that is down.

Data Type

Integer (Boolean)

The example shows a Form_Load that initializes some buttons on a ButtonBar control. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

SpaceAfter Property Array, ButtonBar Control

see also example

Description

Sets or retrieves the spacing between the buttons on a ButtonBar control.

Usage

[*form!*]*ButtonBar1*.**SpaceAfter(I)** [= *setting%*]

Remarks

The SpaceAfter property array allows you to define the spacing between the buttons on the ButtonBar control. It specifies how much space (in pixels) to leave behind a button before the next button is shown. The index in the SpaceAfter array is the same index as in the Picture array. So, if you set SpaceAfter(0) to 4, you define a spacing of 4 pixels after the first button. The default setting is 0 pixels, but you can set it to any value you like.

Data Type

Integer

The example shows a Form_Load that initializes some buttons on a ButtonBar control. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

ShowStatusMessage Property, ButtonBar Control

see also example

Description

Specifies when the message, set in the ButtonMessage property array is displayed in the StatusBar.

Usage

[*form!*]*ButtonBar1*.**ShowStatusMessage** [= *setting%*]

Remarks

It is possible to make a connection between the StatusBar control and the ButtonBar control, by filling in the StatusField and hWndStatusBar properties. After that you can specify a message for each button or control you have defined, by filling in the appropriate entry in the ButtonMessage property array or in the ControlMessage property. The message will be displayed in the specified field of the StatusBar control depending on the setting of the ShowStatusMessage property.

The ShowStatusMessage takes one of the following values to specify when the message is displayed in the StatusBar:

Value	Short name	Description
0	NEVER	(Default) Does not display the messages in the StatusBar.
1	MOUSESELECT	Displays a message in the StatusBar when the button is selected and clear the message after the button is released. Does not display ControlMessages since I still havent figured out how to trap these without using a Windows-hook.
2	MOUSEOVER	Displays a message in the StatusBar when the mouse is over a button or control on the ButtonBar. The message is cleared if the mouse moves off the button or control.
3	RIGHTMOUSE	Displays a message in the StatusBar when the button is selected with the right mouse-button. The message is cleared when the right mouse-button is released. Does not display control-messages.
4	WITHHINT	Display the Button/ControlMessages along with the hint-popup. The message is placed in the StatusBar only if the ShowHint property is set to True.

Data Type

Integer (Enumerated)

The example shows a Form_Load that initializes some buttons with messages on a ButtonBar control. To use this example create a form with one ButtonBar and one StatusBar control and paste the code into the Declarations section of your form.

ButtonMessage Property Array, ButtonBar Control

see also example

Description

Sets or retrieves the text to be displayed in one of the text-fields of a StatusBar control if the button is selected.

Usage

[*form!*]*ButtonBar1*.**ButtonMessage(I)** [= *message$*]

Remarks

It is possible to make a connection between the StatusBar control and the ButtonBar control, by filling in the StatusField and hWndStatusBar properties.

After that you can specify a message for each button you have defined, by filling in the appropriate entry in the ButtonMessage property array. The message will be displayed in the specified field of the StatusBar control depending on the setting of the ShowStatusMessage property.

Data Type

String

The example shows a Form_Load that initializes some buttons with messages on a ButtonBar control. To use this example create a form with one ButtonBar and one StatusBar control and paste the code into the Declarations section of your form.

ButtonGroup Property Array, ButtonBar Control

see also example

Description

Sets or retrieves the group a button on the ButtonBar control is a member of.

Usage

[*form!*]*ButtonBar1*.**ButtonGroup(I)** [= *setting%*]

Remarks

You can group 2-state buttons in a group, by setting the ButtonGroup property of all the buttons you want to group to the same number.

When buttons are selected into a group, they behave like toggles; when one of the buttons in a group is depressed, all other buttons are raised. Once a button is depressed it cannot be raised again by clicking on it a second time, unless you alter the GroupAllowAllUp property for the group the button is a member of.

To remove a button from a group, set the ButtonGroup property of that button to -1.

Data Type

Integer

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

LeftMargin Property, ButtonBar Control

Description

Sets or retrieves the left-margin for the ButtonBar control.

Usage

[*form!*]*ButtonBar1*.**LeftMargin** [= *setting%*]

Remarks

The LeftMargin property specifies how much room to leave before displaying the first button

The left margin is always expressed in pixels, regardless of the ScaleMode of the ButtonBar's parent.

Data Type

Integer

ButtonPicture Property, ButtonBar Control

see also example

Description

A one-dimensional array of pictures representing all of the picture cells in the bitmap specified by the ButtonPictures property. This property is not available at design time and is read-only at run time.

Usage

[*form!*]*ButtonBar1*.**ButtonPicture**(Index%)

Remarks

Use the MKBitmap program provided with this VBX to create a bitmap that contains all the pictures needed to represent the different states of each button on the ButtonBar.
Use the ButtonPictures property to specfiy the created bitmap for use with the ButtonBar.
Use the ButtonRows and ButtonColumns properties to divide the bitmap into a uniform matrix of graphic cells.
Use this property to specify a picture for the Picture , PictureDown , PictureDisabled or PictureDisabledDown property arrays.
The cells specified by ButtonPicture are indexed, beginning with 0, and increase from left to right and top to bottom.
When reading this property, an error is generated when there is no picture or the Rows or Cols property is set to 0.

Integer (Picture)

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control using the ButtonPictures property. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

Insert button

If you click this button, a new button is inserted before the currently selected button. If you already defined 30 buttons, the last button is moved off the ButtonBar. All buttons, including the currently selected button are moved one place down, to accomodate space for the new button.

Delete button

If you click this button, the currently selected button is deleted. All buttons after this button are moved one place up. If you already defined 30 buttons, the last button will become available again.

Up button

This button (together with the down-button) allows you to exchange buttons. If you click this button, the currently selected button is exchanged with the button right before it, effectively moving it one place up.

Down button

This button (together with the up-button) allows you to exchange buttons. If you click this button, the currently selected button is exchanged with the button after it, effectively moving it one place down.

Cancel button

To ignore the changes you've made for the settings for the buttons on the ButtonBar control, click the Cancel-button. The ButtonBar will remain exactly the same as it was before you selected the ButtonProperties property.

Ok button

To confirm the settings for the buttons on the ButtonBar control, click the Ok-button.

All settings you've specified will be applied to the ButtonBar control. Now you can see directly how your ButtonBar will look when the program is run.

Bitmap selection scroll-bar

The bitmap you've specified in the ButtonPictures property is broken into little bitmaps according to the settings in the ButtonColumns and ButtonRows properties. You can select one of the tiny bitmaps for use on a button on the ButtonBar by sliding the scroll-bar to the left or to the right until you find the bitmap you need.

To use the selected bitmap for one of the states of the currently selected button, press the Use-button.

You can also select one of the tiny bitmaps at run-time by requesting a bitmap from the ButtonPicture (graphic cells) array.

StatusBar Field textbox

Use this property to set the field of the StatusBar in which the messages in the ButtonMessage property array are to be displayed. Valid values range from 0 to 29 inclusive. You can also set this property at run-time by setting the StatusField property.

StatusBar selection combo-box

This combobox present a list of all StatusBar available on the current form for displaying the messages in the ButtonMessage property array. To connect a StatusBar to the current ButtonBar, select its name from the combobox list. To remove the connected between a StatusBar and the ButtonBar, select <none> from the list. You can also set this property at run-time bu setting the hWndStatusBar property.

Select button

Clicking this button will bring up the dialog where you can select a bitmap that has to be used for representing the selected state of the currently selected button. This allows you to use bitmaps that are not present in the large bitmap in the ButtonPictures property.

Use button

After you've selected the tiny bitmap you want to use, and the state of the button this picture is for, clicking the Use-button will copy the picture to the selected position for the cuurently selected button.

Picture selection radio buttons

After you've selected the tiny bitmap you want to use, select the state of the button this picture is for by clicking on one of the radiobuttons. Click on the 'Use'-button to copy the picture to the selected location.

To remove a button-picture for a button, double click on it.

You can also set the pictures for a button by setting the appropriate entries in the Picture , PictureDown , PictureDisabled or PictureDisabledDown property arrays at run-time.

ButtonHint

Allows you to set a hint that is displayed over the button when the mouse cursor rests on it.

You can also set this property by setting the appropriate entry in the ButtonHint property array at run-time.

ButtonTag

Allows you to set a text specific for a button, for example to be able to identify the button in code, even if you change the order of the buttons.

You can also set this property by setting the appropriate entry in the ButtonTag property array at run-time.

ButtonMessage

Allows you to set a message that can be displayed in one of the text-fields of a StatusBar control.

You can also set this property by setting the appropriate entry in the ButtonMessage property array at run-time.

GroupAllowAllUp Checkbox

If this check-box is checked the group the currently selected button is part of, allows all buttons in the up-position. If this check-box is unchecked at least one button of the group must be down.

You can also set this property by specifying a value for the GroupAllowAllUp property at run-time.

ButtonVisible Checkbox

If this check-box is checked the currently selected button is visible, otherwise the button is invisible.

You can also set this property by specifying a value for the ButtonVisible property at run-time.

ButtonEnabled Checkbox

If this check-box is checked the currently selected button is enabled, otherwise the button is disabled.

You can also set this property by specifying a value for the ButtonEnabled property at run-time.

ButtonGroup

The ButtonGroup text-box array allows you to make 2-state buttons part of a group. To make buttons part of the same group assign the same group number to them. Legal groupnumbers run from 0 to 29 inclusive.

You can also set this property by setting the appropriate entry in the ButtonGroup property array at run-time.

Space After

The SpaceAfter text-box array allows you to define the spacing between the buttons on the ButtonBar control. It specifies how much space (in pixels) to leave behind the currently selected button before the next button is shown. So if you set the value to 4, you define a spacing of 4 pixels after the current button. The default setting is 0 pixels, but you can set it to any value you like.

You can also set this property by setting the appropriate entry in the SpaceAfter property array at run-time.

ButtonType selection combo-box

Select the type of button for the currently selected button. Possible values are:

Value	Description
Normal Button	(Default) The button acts like a normal command-button.
2 State Button	The button is a 2-state button. After the first click, the button remains depressed. After the second click the button returns to it's normal state.

You can also set this property by setting the appropriate entry in the ButtonType property array at run-time.

ButtonState selection combo-box

Select the state for the current button if this button is defined as a 2-state button. Possible values are:

Value	Description
Up	(Default) The button is in the up-position
Down	The button is down.

You can also set this property by setting the appropriate entry in the ButtonState property array at run-time.

Button Listbox

Choose the button you want to specify from this listbox. When you select a button from the list, it's settings are displayed in the combo-boxes and text-boxes to the right and below the listbox.

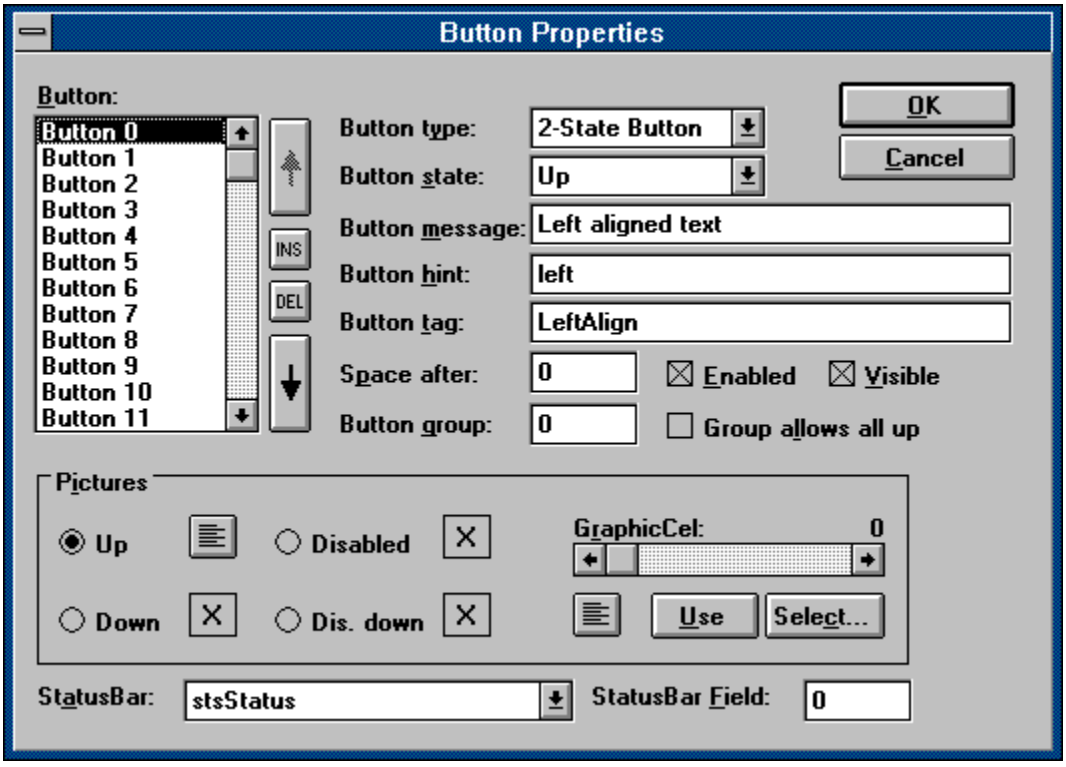
ButtonProperties Dialog, ButtonBar Control

Description

Allows you to set the properties for all button on the ButtonBar at design time, using a dialog-window.

Usage

Double click on the three dots next to the property to display the Button-Properties Dialog. The dialog looks like this. To get help on the way to use the dialog, click the various parts of it for detailed information.



ButtonPictures Property, ButtonBar Control

see also example

Description

This property is the same as the standard Visual Basic Picture property except that it only supports bitmap (.BMP) files.

Usage

[*form!*]*ButtonBar1*.**ButtonPictures** [= *LoadPicture("FileName")*]

Remarks

Use this property to specify a bitmap that contains all the different bitmaps needed for the buttons on the ButtonBar. You can divide the source bitmap specified in the ButtonPictures property into a uniform matrix of picture cells by using the ButtonRows and ButtonColumns properties. Use the ButtonPicture property to specify individual pictures for use with the Picture , PictureDown , PictureDisabled or PictureDisabledDown property arrays.

Data Type

Integer (Picture)

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control using the ButtonPictures property. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

ButtonTag Property Array, ButtonBar Control

see also example

Description

Specifies a tag for a button of the ButtonBar control.

Usage

[*form!*]*ButtonBar1*.**ButtonTag(I)** = *String$*

Remarks

Each button on the ButtonBar can have a tag that identifies the button. Since the ButtonTag property is not used internally in the ButtonBar control, you can use the ButtonTag property to do anything you like, for example giving each button a unique name. This way, you can identify a button easily in the ButtonClick event, without having to rewrite code if you change the order of the buttons.

Data Type

String

The example shows a Form_Load that initializes some buttons on a ButtonBar control and uses the ButtonTag property to identify the buttons in the Click event. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

ButtonState Property Array, ButtonBar Control

see also example

Description

Specifies the state for a button of the ButtonBar control.

Usage

[*form!*]*ButtonBar1*.**ButtonState(I)** = *State%*

Remarks

Each 2-state button on the ButtonBar can have the state RAISED (0) or DEPRESSED (-1). The ButtonState property array takes one of the following values to specify the state of a text-field.

Value	Short name	Description
0	RAISED	(Default) The button is raised.
1	DEPRESSED	The button is depressed.

It is not possible to change the state of a normal button (ButtonType = 0) on the ButtonBar. Using the ButtonState property allows the programmer to reset a 2-state button to it's up-state even if the GroupAllowAllUp property of the group is set to False.

Data Type

Integer (Enumerated)

The example shows a Form_Load that initializes some buttons on a ButtonBar control. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

ButtonVisible Property Array, ButtonBar Control

see also

Description

Sets or retrieves the visibility for the the button with the same index.

Usage

[*form!*]*ButtonBar1*.**ButtonVisible(I)** [= *setting%*]

Remarks

The ButtonBar allows you to hide and show buttons at run-time. You can do this by setting the appropriate entry in the ButtonVisible property array.

There are two ways the ButtonBar can respond if you change the visibility of a button, depending on the setting of the IgnoreInvisibleButtons property. If this property is set to True and you hide a button on the ButtonBar, the rest of the buttons will shift to the left. If you make the button visible again, the rest of the buttons will shift back to the right. If the IgnoreInvisibleButtons property is set to False, only the specified button will disappear; other buttons are not affected. This way you effectively create a gap in the ButtonBar.

The index in the ButtonVisible property array is the same index in the all the other button-related property arrays (i.e. ButtonState, ButtonGroup, ButtonEnabled, Picture).

Data Type

Integer (Boolean)

IgnoreInvisibleButtons Property, ButtonBar Control

see also

Description

Sets or retrieves the way other buttons react to invisible buttons.

Usage

[*form!*]*ButtonBar1*.**IgnoreInvisibleButtons** [= *setting%*]

Remarks

The ButtonBar supports two different button-types. The ButtonType property array takes one of the following values to specify the type of a button.

Value	Description
False	(Default) If you hide a button on the ButtonBar, the rest of the buttons will shift to the left filling up the space the invisible button left behind. If you make the button visible again, the rest of the buttons will shift back to the right.
True	If you hide a button on the ButtonBar by setting its ButtonVisible to False, only the specified button will disappear; other buttons are not affected. This way you effectively create a gap in the ButtonBar.

Data Type

Integer (Boolean)

ButtonType Property Array, ButtonBar Control

see also example

Description

Sets or retrieves the button-type for the button with the same index.

Usage

[*form!*]*ButtonBar1*.**ButtonType(I)** [= *ButtonType%*]

Remarks

The ButtonBar supports two different button-types. The ButtonType property array takes one of the following values to specify the type of a button.

Value	Short name	Description
0	BUTTON_NORMAL	(Default) The button reacts like a normal command-button.
1	BUTTON_2STATE	The button is a 2-state button. After the first click, the button remains depressed. After the second click the button returns to it's normal state. This behavior can be different if the A 2-state button is member of a group (by setting the ButtonGroup property). If the button is part of a group, it is raised if another member of the group is depressed. If the GroupAllowAllUp property of the group the button is a member of is set to False, the button remains depressed if you click on it a second time.

The index in the ButtonType property array is the same index in all the other button-related property arrays (i.e. ButtonState, ButtonGroup, ButtonEnabled, Picture).

Data Type

Integer (Enumerated)

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

ButtonColumns, ButtonRows Properties, ButtonBar Control

see also example

Description

Sets or returns the total number of columns or rows in the picture specified in the ButtonPictures property.

Usage

[*form!*]*ButtonBar1*.**ButtonColumns** [= *cols%*]

[*form!*]*ButtonBar1*.**ButtonRows** [= *rows%*]

Remarks

Use these properties to divide the source bitmap specified in the ButtonPictures property into a uniform matrix of picture cells. Use the ButtonPicture property to specify individual pictures for use with the Picture , PictureDown , PictureDisabled or PictureDisabledDown property arrays.

The ButtonBar control must have at least one column and one row.

The height of each picture cell is determined by dividing the height of the source bitmap by the number of specified rows. Leftover pixels at the bottom of the source bitmap (caused by integer rounding) are clipped.

The width of each picture cell is determined by dividing the width of the source bitmap by the number of specified columns. Leftover pixels at the right of the source bitmap (caused by integer rounding) are clipped.

Data Type

Integer

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control using the ButtonPictures property. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

ButtonEnabled Property Array, ButtonBar Control

see also example

Description

Sets or retrieves the enabled-state for the button with the same index.

Usage

[*form!*]*ButtonBar1*.**ButtonEnabled(I)** [= *setting%*]

Remarks

The ButtonEnabled property settings are:

Setting	Description
True	(Default) Allows the button to respond to events.
False	Prevents the object from responding to events.

This property allows buttons to be enabled or disabled at run time. For example, you can disable buttons that don't apply to the current state of the application. When a button is disabled, the picture, specified by the Picture property, will be changed so that it looks disabled (all black pixels will be transformed to dark-gray) if no picture for the disabled state is specified.

Data Type

Integer (Boolean)

The example shows a Form_Load that initializes a couple of buttons on a ButtonBar control. To use this example create a form with one ButtonBar control and paste the code into the Declarations section of your form.

Events

The ButtonBar control supports the following events:

Click DragDrop DragOver

Methods

The ButtonBar control supports the following methods:

Drag	Move	Refresh	ZOrder

Properties

All the properties that apply to the ButtonBar control are listed in the following table. All properties that are marked with an asterisk (*) are only available at run-time.

Align	ButtonColumns	ButtonEnabled *	ButtonGroup *
ButtonHint *	ButtonMessage *	ButtonPicture	ButtonPictures
ButtonProperties	ButtonRows	ButtonState *	ButtonTag *
ButtonType *	ButtonVisible *	ControlHint *	ControlHwnd *
ControlMessage *	Enabled	FontBold	FontItalic
FontName	FontSize	FontStrikethru	FontUnderline
GroupAllowAllUp *	Height	HintBackColor	HintDelay
HintOffsetX	HintOffsetY	HintPosition	hWnd
hWndStatusBar *	IgnoreInvisibleButtons	Index	Left
LeftMargin	Name	OutlineChildren	Parent
Picture *	PictureDown *	PictureDisabled *	PictureDisabledDown *
ShowDisabledHints	ShowDisabledMessages	ShowHints	ShowStatusMessage
SpaceAfter *	StatusField *	Top	Visible
Width			

Note the ButtonEnabled, ButtonGroup, ButtonHint, ButtonMessage, ButtonPicture, ButtonState, ButtonTag, ButtonType, ButtonVisible, ControlHint, ControlHwnd, ControlMessage, GroupAllowAllUp, hWndStatusBar, Picture, PictureDown, PictureDisabled, PictureDisabledDown, SpaceAfter and StatusField properties are only available at run-time. These properties can also be set at design time using the ButtonProperties dialog-box. Name is the default property for the ButtonBar control.

The ButtonBar Custom Control

Properties Methods Events

Description

The Microsoft Visual Basic programming system for Windows comes with a large set of 3D controls. Unfortunately, a 3D button-bar, as used in almost every MS-Windows application is lacking. Therefore the ButtonBar Custom Control was designed. This control allows you to create a very versatile button bar for all your applications.

File Name

TOOLBARS.VBX

Object Type

ButtonBar

Toolbox Icon

Remarks

The ButtonBar allows an application to display a button-bar at the top of a form. The button-bar has 30 fully configurable buttons, which can be completely defined at design time. To get the ButtonBar working, the only thing you have to do is specify a bitmap for the up-position of each button. The ButtonBar control automatically creates the different bitmaps for the down and the two disabled states (up and down) of the button. Like all applications today, the ButtonBar supports tool-tips, a small window that pops up when the user rests the mouse-cursor on a button or a control placed on the ButtonBar that explains the function of the button or control (small hint). The ButtonBar calls these hints Button/ControlHints. It is also possible to connect the button-bar to the status-bar and specify (longer) messages for each button to be shown when that button is selected.

The ButtonBar also offers a PicClip like property 'ButtonPictures'. With this you can specify a bitmap that contains all the small bitmaps for the buttons on the ButtonBar. This avoids having to distribute all the small bitmaps that make up the ButtonBar since the bitmap is saved within your application. A little application (written in Visual Basic) that is distributed with the TOOLBARS.VBX allows easy creation and modification of such a large bitmap.

Usage

To use the ButtonBar, perform the following steps:

1. Add the Toolbars custom control to your project. The StatusBar and the ButtonBar icons will appear in the Visual Basic tool-palette.

2. Add a ButtonBar control to your form by double clicking on the icon in the ToolBox..

3. Define the buttons for the ButtonBar using the ButtonProperties dialog or set the Picture (and eventually the PictureDisabled) and ButtonType properties for the number of buttons you want to display in the Form_Load event procedure of your form.

4. Add code to respond to a Click on a certain button in the ButtonBar.

Distribution Note When you create and distribute applications that use the ButtonBar control, you should install the file TOOLBARS.VBX in the customer's Microsoft Windows \ SYSTEM subdirectory.