Pizazz 1.55

Custom Control Library

Introduction

Welcome to Pizazz 1.55, a Custom Control Library from Visual Bits.

Pizazz is a custom control for Visual Basic 3.0 that adds both a graphical control and a container control to your tool box.

PZLabel - a Visual Basic graphical control

PZPanel - a Visual Basic container control

Pizazz can replace the label, picture, image, and button standard controls, and the 3-D panel and button professional controls, and can emulate tabs and other controls in an almost infinite variety.

Pizazz is shareware.   For information on recieving a registered copy see the Registration topic.

Click here to see what's new.
Click here for a summary of Pizzaz features.
Click here for information on contacting Visual Bits.

What's new in this version

Buttons and bevels, that's what. The AutoClick property makes it possible to easily implement command, toggle, radio, combo, and tab style buttons. ButtonState can be used to set and return the state of those buttons. You can also EatMouse during lengthly database, multimedia, or other operations to prevent a user from clicking something they shouldn't. You'll notice that's much easier then disabling buttons and you don't get the christmas tree effect.

You can now ShowFocus in a variety of ways. Outline the control, or draw a focus rectangle around the caption, or just bold the caption the way Windows 95 likes to. You can also choose to send a click event when an access key is pressed. Speaking of Windows 95, there are some new BevelShade options like 3D frame amd 3D button that will draw a bevel much like a 95 control. You no longer need to use both an inner and outer bevel which is not only easier but much more flexible.

And for completeness, the KeyDown, KeyPress, and KeyUp events have been added to the PZPanel control. Plus you will notice that a CornerSize of one now works much better.

Features

Pizazz is a custom control for Visual Basic 3.0 that adds both a graphical and a container control to your tool box. Pizazz has many unique features. It can behave like a standard label control and display captions up to 64k long. The caption can be aligned in a variety of ways and a margin can be used to fine tune the placement. Auto sizing and word wrap are supported.

But that's just the start. Backgrounds can be filled with an almost infinite variety of gradients or colors. The gradients can be horizontal, vertical, or diagonal and even transition through the center of the control. Even on 256 color systems a palette can be loaded to get super results. And the more colors a system supports the better the gradients look.

The caption can have a shadow of any color for 3D effects. The shadow can be either a block or drop style to the lower right or upper left and of any size or color. This allows an almost infinite variety of styles, from the simple inset or raised that threed.vbx provides to something big and bold for the best impact.

You can add an outer border that can be a solid color or shadow or a faded color or faded drop shadow of any size and color. A faded border is a border that fades from one color into another - it's actually another type of gradient. This allows for a unique framing effect that can be really neat when combined with another picture or gradient. The faded shadow is a shadow that fades from the color of the corners to the color of the outer border. This creates the feathered shadow look you've always wanted behind all those labels and buttons. Of course, until now, it was too much of a hassle to do anything like that. Not anymore.

There's an outer bevel of any size that supports inset and raised effects and can be a number of shades including the new Windows 95 look. The inner border can also be any color or faded color or filled with the same exact color, gradient, or picture used for the background. An inner bevel totally completes the container options.

Corners of any size can be used to get a unique polygon shape. You can change the corner color to blend the corner into the background. A variety of tab shapes are also supported. A button shape is even supported by the tab property that allows controls to be linked together like VCR controls.

Bitmaps can be added and tiled or streched to get the perfect background texture or picture effect. An additional icon can also be placed anywhere on the control.

If the control is not transparent then painting can be done either directly to the screen or in a flicker free manner. Painting can also be disabled and a control can clip itself from the painting region of another underlying control. This allows graphical controls to be layered in complex arrangements with the minimum of painting possible.

This makes it feasible to build highly graphical interfaces entirely out of graphical (non-windowed) Pizazz controls. Because Pizazz is less then 30k in size the application load time is extremly quick and the resource use is minimal. Pizazz has been used as the exclusive add on control to build several commercial multimedia applications.

Visual Bits

Visual Bits specializes in providing the tools and software development expertise to create state of the art Visual Basic applications.

Use the following addresses if you have any questions or comments about Visual Bits or require support for a registered copy of Pizazz:

Compuserve E-Mail
70402,3651

Internet E-Mail
70402.3651@compuserve.com

Mailing Address
Visual Bits
PO Box 243
Watertown, MA 02272

Pizazz developed for Visual Bits by Ben Jones.

Installation

You should have received the following files:

PIZAZZ.VBX	The custom control file. You may distribute this file with your application.
PIZAZZ.HLP	This help file
README.WRI	Introduction and notes (if any)
PZDEMO.MAK	Visual Basic project file for the demo program
*.FRM	Visual Basic form files for the demo program
*.FRX	Binary data for the above form files
PZDEMO.EXE	Executable demo program
LICENSE.WRI	Legal stuff pertaining to the license if registered.

Registered copies contain the following additional files:

PIZAZZ.LIC	This file, which must reside in the Windows System directory, contains license information. It is needed to create executables that do not display a registration reminder screen. You are not allowed to distribute the PIZAZZ.LIC file with any application that you develop and distribute. The information in the license file can be displayed by selecting the (About) property.
THETIME.ZIP	A very graphical replacement for the window's standard clock. Show's how to use both Pizazz.vbx and Modblast.vbx. All source code included.
MODBLAST.VBX	A modified version of the Msgblast subclass control that comes with the Microsoft Developer Network CD. This is stored in THETIME.ZIP.
BG*.BMP	All background textures used in the Pzdemo program.
PZ*.BMP	1 x 1 bitmaps containing palettes used by Pzdemo.
*.ICO	All icons used in the Pzdemo program.
PAL.ZIP	A palette display custom control (this control is an example taken from the Visual Basic 3.0 CDK).

Copy the files together to any directory.   You should copy Pizazz.vbx, pizazz.hlp and pizazz.lic (if registered) to your windows system directory (usually \windows\system).

Pizazz will always look for the help file in the same directory that the .VBX is loaded from.

Both THETIME.ZIP and PAL.ZIP should be unzipped to seperate directories.   Copy the VBX files into the windows system directory.

Enjoy!

Usual Disclaimer

Pizazz is not public domain or free software.
Pizazz is shareware.   You may use the non-registered copy of Pizazz for evaluation purposes only.

The PIZAZZ.VBX file may be freely distributed with any applications. The non-registered version of Pizazz is fully functional but will occasionally display a registration reminder screen at design time and in executables created by the non-registered version.   The registered copy of Pizazz contains a license file which removes these reminder screens and provides for support via e-mail.

To receive a registered copy,   please follow the instructions in the Registration topic of the Pizazz.hlp help file.   Pizazz can also be registered via the "GO SWREG" forum in Compuserve - enter 6551 for the registration ID.

Important Note!
You are not allowed to distribute the PIZAZZ.LIC file with any application that you develop and distribute.

Disclaimer
This software and documentation are supplied "AS IS".   The author makes no warranty of any kind, either express or implied, with respect to this software and accompanying documentation.

In no event shall the author of this software be liable for any damages arising out of the use of this product.   Your use of this software indicates that you have read and agreed to these terms.

Acknowledgements
Microsoft is a registered trademark of Microsoft Corporation.

Visual Basic, TrueType, and Windows are trademarks of Microsoft Corporation.

Registration

General pricing information:

```
Single License:     $15.00
 2-10 User License: $13.00 per user
11-20 User License: $10.00 per user
Site License       $200.00 per site

Shipping     USA................(Amount is for the       $ 10.00
    &        Canada and Mexico...entire order regardless $ 16.00
 Handling    Other Countries.....of number ordered.)     $ 18.00
```

🟩 **Click here for information on registering by Credit Card.**

🟩 **Click here for Compuserve electronic registrations.**

🟩 **Click here for information on registering by check.**

Registering by Credit Card

If you have a valid Visa or MasterCard, you may register through NorthStar Solutions credit card services. Contact NorthStar via any of the following methods.

```
Voice: 1-800-699-6395 (10am - 10pm, EST.  US calls only)
       1-803-699-6395 (10am - 10pm, EST.)

FAX:   1-803-699-5465 (24 hours. International and business
                 orders encouraged.)

EMAIL: American Online: STARMAIL  Internet: Starmail@aol.com
```

Please have the following information ready:

- The program name you are registering (Pizazz VBX).
- The number of registered copies that you are purchasing.
- Your name (and company name if a business) and mailing address.
- Your Visa or Mastercard # and expiration date.

And please note:

- NorthStar processes registrations only, see the Support topic for support information.
- E-mail and Fax registrations are encouraged, but all registrations are much appreciated!

Compuserve Registration

Pizzaz can be registered via Compuserve. The registration cost is automatically billed to your Compuserve account.   The "GO SWREG" numbers are:

Pizazz Custom Control: (Registration ID) 6551

When registering via Compuserve, the registered product will be distributed to you via Compuserve's electronic mail and there will be no shipping charges   Any taxes, if applicable, will be handled automatically by Compuserve.

Registering by Check or Money Order

To registered by US check or money order drawn from a US bank, please print out the following order form or write down the equivalent by hand, and follow the instructions.

Your address:

```
Name      ___________________________________________________________

Company   ___________________________________________________________

Address   ___________________________________________________________

          ___________________________________________________________

City      ________________________ St./Prov. _______________________

ZIP/Code __________________ Country (if not USA) __________________

Phone/Fax _______________________________

E-Mail Address: _________________________ (Internet/Compuserve)

Where did you find this product? __________________________________
```

Quantity and pricing:

```
Pizazz Custom Control (per user).... $15.00 x ____   =   _________

     2 - 10 users................... $13.00 x ____   =   _________

    11 - 20 users................... $10.00 x ____   =   _________

    Site License (per site).........$200.00 x ____   =   _________

   Sales Tax (MA Residents only - add 5%)...........     _________

Shipping     USA................(Amount is for the       $ 10.00
    &        Canada and Mexico...entire order regardless $ 16.00
 Handling    Other Countries.....of number ordered.)     $ 18.00

Grand Total.........................................     _________
```

Paying by check:

Make checks and money orders payable to: Visual Bits
Payment must be in U.S. dollars and drawn against a U.S. bank.

```
 Mail to:  Visual Bits
           P.O. Box 243
           Watertown, MA  02272-0243
```

Support

Support will be provided by e-mail to registered users.   Support questions will be answered as quickly as possible.

Registered owners are encouraged to provide feedback and suggestions!

See the Visual Bits topic for e-mail and US mailing addresses.

Control Reference

Properties (default Caption):
Properties specific to Pizazz are marked with an asterisk.

Click here to see the specific properties only.
Other standard properties are documented in the Visual Basic help files.

* (About)	* ClipCtl	ForeColor
* Align [1]	* CornerBackColor	* GradientColor
* Alignment	* CornerSize	* GradientStyle
* AutoClick	CtlName	* hCtl
* AutoSize	DataChanged	Height
BackColor	DataField	* Icon
BackStyle [2]	DataSource	* IconLeft
* BevelInner	* DirectDraw	* IconTop
* BevelInnerShading	* DisabledForeColor	Index
* BevelInnerWidth	DragMode	Left
* BevelOuter	DragIcon	MousePointer
* BevelOuterShading	* EatMouse	Name
* BevelOuterWidth	Enabled	Parent
* BorderInner	* EnablePainting	* Picture
* BorderInnerColor	* Font3D	* PictureStyle
* BorderInnerWidth	* Font3DColor	* ShowFocus
* BorderOuter	* Font3DSize	TabIndex
* BorderOuterColor	FontBold	TabStop [1]
* BorderOuterWidth	FontItalic	* TabStyle
* ButtonState	FontName	Tag
Caption	FontSize	Top
* CaptionMargin	FontStrike	Visible
* CaptionPrefix	FontUnder	Width
		WordWrap

Events:
Pizazz supports the following standard events which are documented in the Visual Basic help files.

Change	DragOver	MouseDown
Click	KeyDown [3]	MouseMove
DblClick	KeyPress [3]	MouseUp
DragDrop	KeyUp [3]	

Notes:

[1] - The Align and TabStop properties are only available to the PZPanel control.
[2] - The BackStyle property is only available to the PZLabel control.   This property behaves exactly like the standard BackStyle property except it supports an additional setting of 2 - "Opaque (including corners)" - that paints the CornerBackColor of a PZLabel control.
[3] - These events are not supported by the PZLabel control.

Specific Properties

The following is a list of the properties specific to the Pizazz custom control.

(About)
Alignment
AutoClick
AutoSize
BevelInner, BevelOuter
BevelInnerShading, BevelOuterShading
BevelInnerWidth, BevelOuterWidth
BorderInner, BorderOuter
BorderInnerColor, BorderOuterColor
BorderInnerWidth, BorderOuterWidth
ButtonState
CaptionMargin
CaptionPrefix
ClipCtl
CornerBackColor
CornerSize
DirectDraw
DisabaledForecolor
EatMouse
EnablePainting
Font3D
Font3DColor
Font3DSize
GradientColor
GradientStyle
hCtl
Icon
IconLeft, IconTop
Picture
PictureStyle
ShowFocus
TabStyle

(About) Property

Description

Displays a dialog box indicating the Pizazz version number and registration information.

Usage

Click on the ellipses ('...') button next to the property text to activate the about dialog box.

Remarks

Only available at design time.

Data Type

Popup

Alignment Property

Description

Determines how the caption will be aligned on the control..

Usage

[form.]control.Alignment[= setting]

Remarks

The settings for this property are:

Setting	Description
0	Left Justify - Top
1	Left Justify - Middle
2	Left Justify - Bottom
3	Right Justify - Top
4	Right Justify - Middle
5	Right Justify - Bottom
6	Center - Top
7	(Default) Center - Middle
8	Center - Bottom
9	Center Offset - Top
10	Center Offset - Middle
11	Center Offset - Bottom

The position of the caption is also adjusted by the CaptionMargin property. The caption is drawn inside any inner and outer borders and bevels.

For the Left and Right Justify settings the margin is added to the left or right sides of the caption. For the Center settings the margin is added to both sides. The Center Offset setting adds the margin to the left side only, effectively moving the center point of the caption.

Data Type

Integer (Enumerated)

See Also

CaptionMargin

AutoClick Property

Description
Determines if the control automatically changes the bevels, border shadow, or zorder of a control on a click event.

Usage
[form.]control.AutoClick[= setting]

Remarks
The settings for this property are:

Setting	Description
0	(Default) None. Control will not automatically change any setting.
1	Command. The control changes either a bevel or the outer border shadow on both a left mouse down and a left mouse up.
2	Combo. Works like the Command setting, except other controls within the same container that have this same setting and the same index value are also automatically clicked.
3	Toggle. The control automatically changes the bevel or border shadow setting on a left mouse click.
4	Radio. Similar to a toggle, except other controls within the same container that have this same setting and the same name (must be part of a control array) have their bevels or border shadow set to the off (raised) state.
5	Tab. The control's zorder is set to 0 (front) on a left mouse down. Other controls within the same container that have this same setting and the same name (must be part of a control array) have their zorder set to 1 (back).

All these settings, except for 5 (Tab), work by making the control appear to be raised to represent the off (False) button state or appear to be inset to represent the on (True) button state. The 0 (Command) and 1 (Combo) settings don't set the button state but do generate a click while lowering and raising the control.   If the control has an outer border set to the Shadow or Faded Shadow state, then the shadow size is set to zero and the control is moved to the lower right to cover the shadow when the button state is on. When the button state is off the shadow is exposed.   Otherwise the button state is shown by insetting or rasing either the inner or the outer bevel.   The inner bevel is used if it isn't set to 0 (None) and has a width of two or greater.   Otherwise the outer bevel is used.

The ButtonState property is set to -1 (True) for settings 3 (Toggle) and 4 (Radio) when the button state is on.   This property is set to True for setting 5 (Tab) when the control is sent to the front, or False when it is sent to the back of the zorder.   Setting this value to True or False will set the bevel or zorder accordingly.

A click event is sent for 0 (Command) or 1 (Combo) settings to the control that recieved the

MouseUp event.   A click event is sent for Toggle, Radio, and Tab controls whenever the ButtonState value changes, regardless of whether the control recieved any actual mouse events.

Note that the Combo setting is used to combine several controls together that act as one button. This can, for instance, be used to build a button with attached tabs or pictures.   Buttons that are combined together must be within the same container, such as a PZPanel or picture box, have the Combo setting, and have the same index value or have no index value.

The Radio and Tab settings work with controls within the same container that have the same name (so they must be part of a control array).

Data Type
Integer (Enumerated)

See Also
BevelInnerWidth, BevelOuterWidth, BorderOuter, ButtonState

AutoSize Property

Description
Determines whether the control is automatically resized to fit its caption.

Usage
[form.]control.AutoSize[= {True|False}]

Remarks
The settings for this property are:

Setting	Description
True	Automatically resizes the control to fit its caption.
False	(Default) Keeps the size of the control constant. The caption is clipped when it exceeds the area of the control.

If the WordWrap property is False then the control is stretched horizontally. Otherwise it is stretched vertically.

Data Type
Integer (Boolean)

BevelInner, BevelOuter Property

Description
Determines the style of the inner or outer bevel of the control.

Usage
[form.]control.BevelInner[= setting]
[form.]control.BevelOuter[= setting]

Remarks
The settings for this property are:

Setting	Description
0	(Default) None. No bevel is drawn.
1	Inset. The bevel appears inset on the screen.
2	Raised. The bevel appears raised off the screen.

The various styles of bevels and borders can be combined to create a large variety of effects.

Data Type
Integer (Enumerated)

See Also
BevelInnerShading, BevelOuterShading, BevelInnerWidth, BevelOuterWidth, BorderInner, BorderOuter

BevelInnerShading, BevelOuterShading Property

Description
Specifies the type of shading for a inner or outer bevel.

Usage
[form.]control.BevelInnerShading[= setting]
[form.]control.BevelOuterShading[= setting]

Remarks
This property is used to determine the shading used by an inner or outer bevel.

The settings for this property are:

Setting	Description
0	(Default) Button. White and dark gray.
1	Light. Light gray and dark gray.
2	Dark. Light gray and black
3	Frame. White and black.
4	Outlined Button. Black outline combined with button and light shading.
5	3D Button. Frame combined with button and light shading.
6	3D Frame. Dark combined with button shading.

Note: the 3D Button and 3D Frame shadings correspond closely to the look of equivalent Windows 95 controls.

The Outlined Button, 3D Button and 3D Frame combine two shades together to draw the bevel.   The outside shade is always one pixel wide.   The size of the inner shade is the rest of the bevel width. The button combinations use the button shade when raised and the light shade when inset.

Data Type
Integer (Enumerated)

See Also
BevelInner, BevelOuter, BevelInnerWidth, BevelOuterWidth

BevelInnerWidth, BevelOuterWidth Property

Description

Sets or returns the width of the outer or inner bevels of the panel; determines the amount of the three-dimensional shadow effect.

Usage

[form.]control.BevelInnerWidth[= width]
[form.]control.BevelOuterWidth[= width]

Remarks

The setting for this property determines the number of pixels used to draw the inner or outer bevels that surround the control..

Bevel width can be set to a value between 0 and 30, inclusive.

Data Type

Integer

See Also

BevelInner, BevelOuter, BevelInnerShading, BevelOuterShading

BorderInner, BorderOuter Property

Description
Determines the style of the inner or outer border of the control.

Usage
[form.]control.BorderInner[= setting]
[form.]control.BorderOuter[= setting]

Remarks
The settings for this property are:

Setting	Description
0	(Default) None. No border is drawn.
1	Solid. The border is drawn as a solid color.
2	Filled. The border is filled with the same color or bitmap as the background.
3	Shadow. The border appears as a lower right drop shadow.
4	Faded. The border color fades from one color to another.
5	Faded Shadow. The border is a lower right faded drop shadow.

The Shadow and Faded Shadow styles are not supported by the inner border. The Filled style is not supported by the outer border.

Both the Solid and Shadow style colors are determined by the border color settings. The Faded Shadow style fades from the CornerBackColor to the border color. The outer border Faded style fades from the outer border color to the inner border color. The inner border Faded style fades from the inner border color to the BackColor.

The Faded Shadow style can be used to produce a faded drop shadow. This effect is especially appealing when the corner color matches the back ground color. Corner's do not need to be painted to achieve this look. The Faded style is well suited for a unique framing effect.

The outer border is drawn outside of the outer bevel, if any. The inner border is drawn between the inner and outer bevels, if any.

The various styles of bevels and borders can be combined to create a large variety of effects.

Data Type
Integer (Enumerated)

See Also
BorderInnerColor, BorderOuterColor, BorderInnerWidth, BorderOuterWidth, BevelInner, BevelOuter

BorderInnerColor, BorderOuterColor Property

Description
Determines the color of a solid inner or outer border.

Usage
[form.]control.BorderInnerColor[= color]
[form.]control.BorderOuterColor[= color]

Remarks
This property only takes effect when the corresponding inner or outer border is set to 1 (Solid).

Data Type
Long (color)

See Also
BorderInner, BorderOuter, BorderInnerWidth, BorderOuterWidth

BorderInnerWidth, BorderOuterWidth Property

Description
Sets or returns the width of the inner or outer border.

Usage
[form.]control.BorderInnerWidth[= width]
[form.]control.BorderOuterWidth[= width]

Remarks
The setting for this property determines the number of pixels used to draw the inner or outer borders.

The width of the inner border determines the seperation of the inner and outer bevels.

The border width can be set to a value between 0 and 30, inclusive.

Data Type
Integer

See Also
BorderInner, BorderOuter, BorderInnerColor, BorderOuterColor, BevelInner, BevelOuter

ButtonState Property

Description

Determines and returns the button state of a control.

Usage

[form.]control.ButtonState[=   {True|False}]

Remarks

This property has no effect unless the AutoClick property is set to 3 (Toggle) or higher.

After a control recieves a click event, this property can be used to determine the state of the button or tab.

Setting	Description
False	(Default) Button or tab is in the "off" state.
True	Button or tab is in the "on" state.

Data Type

Integer (Boolean)

CaptionMargin Property

Description
Sets or returns the margin used to position the caption on the control.

Usage
[form.]control.CaptionMargin[= margin]

Remarks
The caption is positioned using this property and the Alignment property. The caption is drawn inside any inner and outer borders and bevels.

Use a positive margin to put space between the caption and the left or right edge of the innermost bevel or border.

A negative margin can be used to clip part or all of the caption. The caption can be scrolled by continuously increasing or decreasing the margin.

Data Type
Integer

See Also
Alignment

CaptionPrefix Property

Description
Determines whether support is provided for an access key to a caption prefix.

Usage
[form.]control.CaptionPrefix[= {True|False}]

Remarks
The settings for this property are:

Setting	**Description**
`True`	`Caption prefix and access key is supported.`
`False`	`(Default) No caption prefix is supported.`

If this property is true then you can use the Caption property to assign an access key to the control. In the caption, include an ampersand (&) immediately preceding the character (prefix) you want for an access key. The prefix character will be underlined. Press Alt plus the underlined character to move the focus to that control.

If the ShowFocus property has the 8 bit set then a click event will be generated by the access key. This property can also be used to show the focus in a variety of ways.

Note that a PZLabel can not recieve the focus. However it can still generate a click event.

Data Type
Integer (Boolean)

See Also
ShowFocus

ClipCtl Property

Description
Causes an underlying control to be clipped by this control.   Write only at run time and not available at design time.

Usage
[form.]control.ClipCtl[= [form.]clipping_control.hCtl]

Remarks
Setting this property with the hCtl property of another Pizazz control clips the rectangular region of this control from the painting region of the control referenced by hCtl.

Setting this property to zero or to a new hCtl value removes the previous clipping region.   This property should not be set to any other values.

When a control is unloaded and this property has been set, the control automatically removes the clipping region.

This property is useful when layering PZLabel graphical controls on top of other PZLabel or PZPanel controls that occasionally repaint their client area.   Normally the top-most PZLabel will be repainted also, even if its client area has not changed.   Setting this property to the underlying hCtl will prevent this from occuring.   Since clipping regions have an impact on resources this property should be used only when necessary.

Data Type
Long

See Also
EnablePainting

CornerBackColor Property

Description
Determines the color of the background area that is visible around the corners.

Usage
[form.]control.CornerBackColor[= color]

Remarks
This property only takes effect when the CornerSize property is not zero.

This property is normally used to match the color of the area not covered by the corners with the background color of the container control or form.

The PZLabel control only uses this property when the BackStyle setting is 2 "Opaque (including corners)".   This setting is normally required if the control is inside a form or picturebox.   Otherwise the corners of a PZLabel are not painted which may be prefered when the control is contained by a PZPanel.

Data Type
Long

See Also
CornerSize

CornerSize Property

Description
Sets or returns the size of the corners of the control.

Usage
[form.]control.CornerSize[= size]

Remarks
The setting for this property determines the number of pixels used to draw the corners.

The corners can be used to create a slightly rounded look, a polygon shape, or a diamond shape.

The background color of the area not covered by the corners is determined by the CornerBackColor property of the control.

If this property is changed after a PZLabel control with a BackStyle setting of 1 is initially drawn then the corner background may not be repainted.   Refreshing the form or container control or setting the BackStyle to 2 will correct the problem.

Data Type
Integer

See Also
CornerBackColor

DirectDraw Property

Description
Determines whether the control paints directly to the screen.

Usage
[form.]control.DirectDraw[=   {True|False}]

Remarks
Use this property to paint the control directly to the screen.

Painting directly to the screen is faster then painting to a memory image and sending the image to the screen.   However painting directly to the screen can result in the appearance of flickering in controls that are updated often.   To avoid flickering set this value to 0 (FALSE).

Setting	**Description**
`True`	`(Default) Control is painted directly to the screen.`
`False`	`Painting is done to memory first.`

Data Type
Integer (Boolean)

DisabledForeColor Property

Description
Determines the foreground color used to display the caption when the control is not enabled.

Usage
[form.]control.DisabledForeColor[= color]

Remarks
This property only takes effect when the control is not enabled.

The default for this property is the COLOR_GRAYTEXT system color.

Use this property to give disabled controls a better visual effect then the default grayed text.

Data Type
Long

EatMouse Property

Description
Determines whether mouse events are eaten (captured and suppressed) by the control.

Usage
[form.]control.EatMouse[=   {True|False}]

Remarks
This property is not accessable at design time.

Setting this property to True will capture the mouse and suppress any further mouse events.   Setting this property to False releases the mouse capture.

If a control has the mouse captured and is unloaded then the capture is automatically released.

This property is useful for suppressing mouse events in situations where a lengthly operation is taking place, such as a database query or loading a multimedia file.   Usually it is undesirable to allow the user to click any other controls until the operation is complete.

Setting	Description
`False`	`(Default) Release mouse capture and allow mouse events.`
`True`	`Capture the mouse and suppress mouse events.`

Data Type
Integer (Boolean)

EnablePainting Property

Description
Determines whether the control can paint.   Not available at design time.

Usage
[form.]control.EnablePainting[=   {True|False}]

Remarks
Use this property to temporarily disable painting at run time.

This is useful when it is desirable to change many properties of a control and delay the painting to a later time.

This also can be used to prevent unnecessary background painting that can occur when a graphical control's enabled or zorder property is changed.   If the container is a PZPanel then this property can be set to FALSE before changing the graphical control's property.

The ClipCtl property can also be used to permanently clip a control from a PZPanel's painting region, but if many controls are being clipped it may be more efficient to temporarily disable painting using this property.

Data Type
Integer (Boolean)

See Also
ClipCtl

Font3D Property

Description

Determines the type of 3D shadow displayed behind the caption.

Usage

[form.]control.Font3D[= setting]

Remarks

The settings for this property are:

Setting	Description
0	(Default) None. No 3D shadow is drawn.
1	Block Left. A block style 3D shadow is drawn to the left.
2	Block Right. A block style 3D shadow is drawn to the right.
3	Drop Left. A drop style 3D shadow is drawn to the left.
4	Drop Right. A drop style 3D shadow is drawn to the right.

Data Type

Integer (Enumerated)

See Also

Font3DColor, Font3DSize

Font3DColor Property

Description
Determines the color of the 3D shadow displayed behind the caption.

Usage
[form.]control.Font3DColor[= color]

Remarks
This property only takes effect when the corresponding Font3D property is not zero (None).

Data Type
Long (color)

See Also
Font3D, Font3DSize

Font3DSize Property

Description
Determines the size of the 3D shadow displayed behind the caption.

Usage
[form.]control.Font3DSize[= size]

Remarks
The 3D shadow size is determined in pixels and must be between 0 and 30.   This property only takes effect when the corresponding Font3D property is not zero (None).

Data Type
Integer

See Also
Font3D, Font3DColor

GradientColor Property

Description
Determines the color of the gradient used to fill the background.

Usage
[form.]control.GradientColor[= color]

Remarks
This property only takes effect when the GradientStyle property is not zero.

The gradient color starts with the BackColor setting and then blends into this this setting. The gradient will be painted using as many colors as possible to get the best effect.

Data Type
Long

See Also
GradientStyle

GradientStyle Property

Description
Determines how the gradient is displayed.

Usage
[form.]control.GradientStyle[= setting]

Remarks
The settings for this property are:

Setting	**Description**
0	(Default) None.
1	Vertical. Gradient is painted from top to bottom of control.
2	Horizontal. Gradient is painted from left to right of control.
3	Diagonal. Gradient is painted diagonally from left to right of control.
4	None (Unused for now).
5	Vertical Centered. Gradient is painted from top to bottom of control with the gradient color centered in the middle.
6	Horizontal Centered. Gradient is painted from left to right of control with the gradient color centered in the middle.
7	Diagonal Centered. Gradient is painted diagonally from left to right of control with the gradient color centered in the middle.

The gradient starts with the BackColor and ends with the GradientColor. Gradients look best on systems that can support more then 256 colors. However, even with only 256 colors available it is possible to create a palette and load it into the Picture property to greatly improve the rendering of the gradient.

Data Type
Integer (Enumerated)

See Also
GradientColor

hCtl Property

Description
This property returns the control's internal handle.   Read only at run time and not available at design time.

Usage
[form.]clipped_control.ClipCtl[= [form.]control.hCtl]

Remarks
This property is used for setting the ClipCtl property of another Pizazz control.   The control referenced by this property will exclude the retangular region of the clipped control from its painting region.

Data Type
Long

See Also
ClipCtl

Icon Property

Description

Determines an icon that is displayed by the control.

Usage

[form.]control.Icon[= icon]

Remarks

The icon is positioned within any borders and bevels and the position is set by the IconLeft and IconTop properties.

The icon is drawn on top of the background Picture, if any.

Data Type

Integer (icon)

See Also

IconLeft, IconTop

IconLeft, IconTop

Description
Sets or returns the position of the icon.

Usage
[form.]control.IconLeft[= left]
[form.]control.IconTop[= top]

Remarks
The setting for this property determines the position of the icon in pixels within any borders and bevels on the control.

A negative value can be used to move all or part of the icon off the control.   It is possible to animate or scroll the icon by changing this property.

Data Type
Integer

See Also
Icon

Picture Property

Description

Determines a graphic to be displayed in the control.

Usage

[form.]control.Picture[= picture]

Remarks

This property only supports bitmap graphics.

The graphic is drawn under the icon and caption. The graphic is drawn inside any borders and bevels, unless one of the border styles is set to 2 (filled). In that case the bitmap will be drawn within the border as well.

The PictureStyle property modifies how the graphic is displayed.

Data Type

Integer (picture)

See Also

PictureStyle, BorderInner, BorderOuter

PictureStyle Property

Description

Determines how the graphic represented by the Picture property is displayed.

Usage

[form.]control.PictureStyle[= setting]

Remarks

The settings for this property are:

Setting	Description
0	(Default) Normal. Graphic is displayed in the upper-left corner.
1	Centered. Graphic is centered in the control.
2	Stretched. Graphic is stretched to fit the control.
3	Tiled. Graphic is tiled to fill the control.

This property only takes effect when the picture property refers to a valid graphic.

Data Type

Integer (Enumerated)

See Also

Picture

ShowFocus Property

Description
Determines how the control shows that it has the focus and responds to an access key.

Usage
[form.]control.ShowFocus[= setting]

Remarks
The settings for this property can be any sum of the following values:

Value	Description
0	(Default) None. The control will not show that it has the focus.
1	The control will draw an extra solid border with a width of one around the control using the BorderOuterColor setting.
2	The control will draw a dashed focus retangle around the caption.
4	The control will set the FontBold property to true.
8	The control generates a click event when an access key is entered. If the AutoClick setting is greater then 2 (Combo) then the ButtonState property is toggled as well.

Note that a PZLabel, because it is a graphical control, can never recieve or show the focus. However it can still generate a click event when an access (CaptionPrefix) key is entered.

Other values for this property besides those shown above are ignored.

Data Type
Integer

See Also
AutoClick, ButtonState, CaptionPrefix

TabStyle Property

Description
Determines the style of a tab effect.

Usage
[form.]control.TabStyle[= setting]

Remarks
The settings for this property are:

Setting	Description
0	(Default) Normal. No tab effect.
1	Top Tab. Control is drawn like a top facing tab.
2	Right Tab. Control is drawn like a top facing tab.
3	Bottom Tab. Control is drawn like a top facing tab.
4	Left Tab. Control is drawn like a top facing tab.
5	Top Button. Control is drawn like a top facing button.
6	Right Button. Control is drawn like a top facing button.
7	Bottom Button. Control is drawn like a top facing button.
8	Left Button. Control is drawn like a top facing button.
9	No Horz. Side. Control is drawn without horizontal sides.
10	No Vert. Side. Control is drawn without vertical sides.

Tab styles 1-4 work by not drawing the outside corner, outer border, and outer bevel on one side (the opposite side from the tab orientation). The control can then be positioned adjacent to or on top of another control to make the tab look attached. By using the Zorder method you can make a tab appear to be active or inactive. Other properties such as the ForeColor or BackColor can be used to reinforce this effect.

Because there are no limitations on placement, you can achieve effects not possible with most "true" tab controls. The CornerSize property can be used to modify the appearance of the tab. If an inner border or bevel is used then the size of the inner border and the corner size will determine whether the inner border and bevel are entirely visible on all four sides.

Respond to the click event to perform any actions that correspond to selecting the tab.

Tab styles 5-10 are similar to 1-4. However these settings always draw the entire inner border and inner bevel on all four sides and the outside corner is partially drawn on the missing side(s). This is useful for placing controls adjacent to each other to simulate a horizontal or vertical group of buttons or panels. Settings 9 and 10 can then be used to draw the inside controls in such a group.

Data Type
Integer (Enumerated)

See Also
CornerSize, BevelInner, BorderInner, BevelOuter, BorderOuter

Tips, Tricks, and FAQs

This topic will include any tips, tricks, or frequent questions concerning Pizazz and VB and will be updated whenever we feel like it.

Top Ten Reasons for Using Pizazz

10. Bevels, borders, corners, icons, faded drop shadows.

9. 3D captions with adjustable margins, alignment, wordwrap, autosizing.

8. Pictures that can be centered, tiled, or stretched.

7. Gradients, Gradients, Gradients.

6. Flicker free painting.

5. Context sensitive help.

4. 3D buttons and frames that can ShowFocus, AutoClick and EatMouse.

3. It's small (less then 30K) and loads fast.

2. It's cheap (only $15) and easy to get - register via Compuserve.

1. And the number one reason is...

Why should I register my copy of Pizazz?

When you register Pizazz you receive several benefits. First, if you register electronically via compuserve you will receive by e-mail a registered version within a few days (in fact, usually the same day).

The registered version incudes a license file which allows executables to be built which do not occasionally display a registration reminder screen. Some additional texture bitmaps, icons, and palette bitmaps are also included.

You will also automatically receive notices of updates to Pizazz as soon as they are available. Be the first to have the latest features!

And most importantly, registered owners can also obtain support via e-mail. You may ask questions concerning the use of Pizazz, report any problems, and make suggestions. In fact, several properties (Align and Gradients) were added at the request of registered users.

Thanks in advance for registering your copy of Pizazz!

Visual Bits

I'm still getting the reminder message after registering.

After you recieve the registered copy of Pizazz you need to copy Pizazz.vbx, Pizazz.hlp, and Pizazz.lic to the windows system directory, which is usually c:\windows\system.

After copying these files open any projects that use Pizazz and recompile them.   This will remove the reminder screen from the executable.

That should do it!

Upgrading to Pizazz from another control

If you have a form that uses another control such as the 3D SSPanel that is part of threed.vbx and you want to convert the SSPanel to a PZPanel or PZLabel then try editing the form with a text editor to make the changes. To do this, you must save your form in text format. Before opening the form in VB, use a text editor such as notepad (or hopefully something better) and open the form. You can now directly edit the properties of all the controls in the form. For instance, you may have a SSPanel stored in your form like this:

```
   Begin SSPanel Panel3D1
      BackColor        =   &H00C0C0C0&
      BevelInner       =   1  'Inset
      BevelWidth       =   2
      BorderWidth      =   4
      Caption          =   "Blah Blah"
      Font3D           =   1  'Raised w/light shading
      ForeColor        =   &H00800000&
      Height           =   2655
      Left             =   1260
      TabIndex         =   0
      Top              =   1980
      Width            =   4725
   End
```

This can be changed to a PZPanel by editing the control type and a few properties. Many properties are actually shared by both controls and do not need to be changed. The BevelInner and BevelOuter properties are exactly equivalent. The BevelWidth in the SSPanel needs to be changed to BevelInnerWidth and/or BevelOuterWidth. The BorderWidth needs to be changed to either BorderInnerWidth or BorderOuterWidth and BorderInner or BorderOuter needs to be added. The Font3D doesn't actually need to be changed since the values are compatible. Making the neccessary changes yields the following:

```
   Begin PZPanel PZPanel1
      BackColor        =   &H00C0C0C0&
      BevelInner       =   1  'Inset
      BevelOuter       =   2  'Raised
      BorderInner      =   2  'Filled
      BorderInnerWidth=   4
      Caption          =   "Blah Blah"
      Font3D           =   1  'Block Left
      ForeColor        =   &H00800000&
      Height           =   2655
      Left             =   1260
      TabIndex         =   0
      Top              =   1980
      Width            =   4725
  End
```

As you can see there are just a few changes. After saving the file, VB can be used to open the modified form and the form can be viewed. If you get a message about "errors during load, see ..log for details then a property was probably given a bad value. Exit VB without saving and check out the mentioned log file - chances are the mistake will be easy to correct. Repeat this process until the form loads correctly. Using a text editor is also the easiest method for changing a PZPanel to a PZLabel and vice versa since VB does not allow a control's type to be changed without removing it from a form. Once you get comfortable using a text editor to complement VB you will find that many changes are more readily made outside of VB with the text editor.

Simulating a Flood or Status Property

Several users have requested something similar to the FloodType property found in the SSPanel control that comes with threed.vbx.   It is actually fairly simple to simulate that property by using a couple of picture box controls and a little VB code.   Plus you get the advantage of being able to customize this code to suit your needs.   To make this even easier, a form has been included as an example of how to do this.   Just load "Status.frm" and call the routine "UpdateStatus pct" where pct is a value between 0 and 100 that shows a percentage status.   Of course you can cut and past the controls and code into another form and modify it to suit your needs.   That's all!

Can I make 3D spin and option buttons using Pizazz?

But of course. The trick is to use an icon to show the state of the option or the spinner. Icons can be stored in another hidden Pizazz or image control and copied into the visible option or spinner Pizazz control to change the state. In fact, you can use this technique to achieve some simple animations. The options form which is part of the PZdemo program shows how to do this.

When you register Pizazz you will receive all the icons used to create this form to help you get started.

Adding really cool textures using bitmaps

Pizazz makes it easy to add a background texture to a control. First you need to find a nice texture. There are many CD-Roms available that are filled with textures. Online services such as Compuserve offer graphics forums where textures may be found. Most image or photo editor programs such as PhotoShop, PicturePublisher, Corel, or Fractal Painter make it possible to create very cool textures. Kai's Power Tools is an image editor plug in with a texture module.

Try to create or find a texture that is seamless and reasonably small. Because Pizazz can tile bitmaps it's possible to use a texture that's only 4 x 4 pixels wide that still fills an entire background. Usually the larger the bitmap the more interesting the texture is, so you need to experiment with this.

Also try to limit the colors you use, unless you know the target system can support the required colors. The most common graphics driver setup supports 256 colors. This is know as a palette system. When you load a bitmap, it will use some of the available colors in the palette. A 256 or greater color bitmap will use all the available colors. A good compromise is to use several bitmap textures that contain 16 unique colors. You can load several of these textures and still have plenty of colors left in the system palette for other uses.

If you use the same bitmap texture in several controls it takes up less disk space to load it into only one control at design time and set the picture properties between the controls at run time.

As a bonus, when you register Pizazz you will receive several texture bitmaps to help you get started.

How do I create a palette to support a gradient?

You may not need to create a palette. On true color systems that support more then 256 colors Pizazz will use as many colors as possible to create really awesome gradients. On systems that use 256 colors or less Pizazz will let windows dither the colors to paint the gradient if a palette is not loaded. For some gradients using dithered colors will be sufficient.

However to get the best gradients on a 256 color palette based system, you will want to create and load a palette that supports the gradient. Pizazz will then paint the gradient using the best colors from the palette. This can result in gradients that look as good as they do on a true color system.

A discussion of palettes is way beyond the scope of this topic. Chapter 15 "Creating Graphics for Applications" of the Visual Basic 3.0 Programming Guide covers the basics. Pizazz supports palettes in a manner similar to the form and picture box controls, except it does not use graphics methods.

The easiest method for creating a palette is to first design the form on a system that supports between 32K and 16M colors. Most display cards can support these many colors at some resolutions and 32K is usually sufficient. Complete the form and add any gradients that you need. Then capture the form's image using a screen capture tool and place this into an image editor. There are some capture/edit tools which make this very easy to do (I've used ULead's Image Pals in the past).

Once the image is captured convert it into a 256 color palette based bitmap. Most editors will only fill the palette with the needed colors so this shouldn't use all 256 colors. During the conversion you also might want to specify that the palette be optimized to get the best colors. Crop the bitmap down to 1 pixel in size and make that pixel black or something unobtrusive. Save this and you will now have a small bitmap with a palette that supports your gradients.

Repeat this procedure for all forms that use different gradients. Or it might be possible that a single bitmap can contain a palette that support all the forms. Load this bitmap into the Picture property of any control that has a gradient and you should see an immediate (positive) effect. If multiple controls use the same palette then copy the picture properties at run time. Note that loading a palette has no effect on non palette and true color systems.

The PZDemo program provides an example of how to do this, how to test for the number of colors the system supports.

When you register Pizazz you will receive the palette bitmaps used in the PZDemo program as a bonus.

And the number one reason is...

You don't need those other controls!
Where else can you get a VBX that behaves like a label, panel, tab, button, polygon, option button, multimedia control, spinner, or maybe something that hasn't even be invented yet.

Clip

Clipping means preventing a control or window from painting over another control that is "on top" or higher in the zorder. Normally a window will not paint over another window in this manner. However graphical controls will paint over other graphical controls above them, resulting in a "flashing" effect. While this may be acceptable in some cases, it is usually better to prevent this from occuring. Pizazz provides the hClipCtl and hCtl properties to make this possible.

Container Control

A container control has a unique window handle and device context and may contain other controls. Container controls use more resources then graphical controls but are useful for grouping controls together.   Examples of container controls are the picture box and the 3-D panel in the professional version.   PZPanel is the pizazz container control.

Flicker

Most controls "flicker" when they paint their client areas. This is most obvious when a control like a label or panel3D fills a large area of the screen and has a caption that is updated frequently, such as when showing seconds on a digital clock. The control first paints the background which erases the caption and then redraws the caption. You may not always notice the flicker on a fast system, but it's there. Pizazz does it's painting to a memory device context and outputs the result in one bitblt operation to the screen. This totally eliminates flicker when painting the control and makes it possible to use Pizazz for simple animations.

Graphical Control

Graphical controls paint directly on their parent control and have no window handle or device context of their own.   They can not have other controls placed within them.   They use less resources then non-graphical controls.   Examples of graphical controls are the standard label and image controls.   PZLabel is the Pizazz graphical control.