About Property

Description

The **About** property displays the about box for this component. The about box contains version and copyright information. It is only available at design time.

About SHORELINE SOFTWARE

About SHORELINE SOFTWARE

SHORELINE is a developer haven created to provide excellent products and support. Our developers have been creating MS-Windows based applications since 1985 and real life client server applications since 1987. We are bringing our real world experience into the design and development of unique controls. We are a TRANSDOMINION Company and are part of a software family including the PRISM Client Server Group. Our family of companies offer many services that include:

- Component development
- Application development (off-site)
- Multi-media design and development
- Install development
- Technology transfer
- Training

How to purchase any of our products:

You can order directly from SHORELINE at 800-261-9198 or contact your favorite dealer.SHORELINE accepts VISA, Mastercard or the Discover card. Dealer prices may vary.

How to contact SHORELINE SOFTWARE about our products or service:

We welcome all ideas, comments and suggestions. Please let us know what you think of our pricing and support policies. If you have a unique story about our products please share it with us; our developers love to hear how our products are being used!

CompuServe: 70541, 2436
US Mail: 35-31 Talcottville Road, #123
Vernon, CT 06066-4030
Phone: 800-261-9198
Fax: 203-870-5727
Contact: Glenn A. Field

A division of TRANSDOMINION Corporation

Prices and versions are subject to change without notice. Products and company names are generally trademarks or registered trademarks of their respective companies. We are not responsible for typographical errors.

SHORELINE is always looking for additional talent. We even work with several remote developers for our products. If you feel you have what it takes to be part of SHORELINE send us your cv or resume.
Where to send your resume

AddValue Method

Example

Declaration

procedure AddValue (number:**Longint**);

Description

The **AddValue** method is used to increment the current value. This method is useful whenever you want to add an increment to the current value.

Align Property

Example

Description

The **Align** property determines how the controls align within their container (or parent control). These are the possible values:

Value	Meaning
alNone	The component remains where you place it in the form. This is the **default value**.
alTop	The component moves to the top of the form and resizes to fill the width of the form. The height of the component is not affected.
alBottom	The component moves to the bottom of the form and resizes to fill the width of the form. The height of the component is not affected.
alLeft	The component moves to the left side of the form and resizes to fill the height of the form. The width of the component is not affected.
alRight	The component moves to the right side of the form and resizes to fill the height of the form. The width of the component is not affected.
alClient	The component resizes to fill the client area of a form. If a component already occupies part of the client area, the component resizes to fit within the remaining client area.

If the form or a component containing other components is resized, the components realign within the form or control.

Using the Align property is useful when you want a control to stay in one position on the form, even if the size of the form changes. For example, you could use a panel component with various controls on it as a tool palette. By changing the Align to alLeft, you guarantee that the tool palette always remains on the left side of the form, and always equals the client height of the form.

NOTE: This property follows the standard Delphi Align property found in the installed Borland VCL library.

BackColor Property

See Also Example

Declaration

property BackColor: TColor;

Description

The **BackColor** property is used to set the background color of the progress bar. Even when using a bitmap or LED for the foreground, the selected backcolor is still used. The BackColor property uses the same values as the Color property.

BackgroundColor Property

See Also

NOTE: This property is here for reference purposes only. It is not available for TProgBar. See BackColor property for TProgBar.

Declaration

property BackgroundColor: TColor;

Description

The **BackgroundColor** property determines the background color of the tab set control. The background area of the tab set control is the area between the tabs and the border of the control. For a list of possible color values, see the Color property.

BarShape Property

See Also Example

Declaration

property BarShape: TPBarShape;

Description

The **BarShape** property allows you to change the shape of your Progress Bar. There are two predefined shapes for your bar: (Rectangular, Trapezoidal).

The Direction, TrapezoidalDir, TrapezoidalMin and TrapezoidalShape properties will modify the trapezoidal shape selection.

BarType Property

See Also Example

Declaration

property BarType: TPBarType;

Description

The **BarType** property adjusts the progress bar's appearance from 3D, LED, or Normal. The 3D appearance is similar to the window style introduced in Windows 3.1. The LED type supports multi-colored segments for progress display. Finally the normal type is the standard percent bar appearance.

The Segment properties are used when the LED bartype has been selected. For 3D and Normal, the various Segment properties are ignored.

BevelType Property

See Also Example

Declaration

property BevelType : TBevelStyle;

Description

The **BevelType** allows you to change the appearance of the border region. There are three options available for **BevelType** (Lowered, None, Raised)

BevelWidth Property

See Also Example

Declaration

property BevelWidth : Byte;

Description

The **BevelWidth** property determines the width, in pixels, between the inner and outer bevels of TProgBar.

Bitmap Property

See Also Example

Declaration

property Bitmap : TBitmap;

Description

The **Bitmap** property for TProgBar allows you to use an actual bitmap for the foreground of the progress bar. This property is useful if you want to have a scrolling bitmap message or logo. With this option your opportunities are endless for a customized progress bar look.

BorderType Property

See Also Example

Declaration

property BorderType : TBorderStyle

Description

The **BorderType** property establishes a border surrounding TProgBar. You have two options (None, Single) for this property. A Single border will place a thin, black border around the ProgressBar.

BrushStyle Property

Declaration

property BrushStyle: TBrushStyle

Description

The **BrushStyle** property determines the brush style used for filling the background of the progress bar. You can choose from several different brush styles.

Color

These are the possible values of Color:

Value	Meaning
clBlack	Black
clMaroon	Maroon
clGreen	Green
clOlive	Olive green
clNavy	Navy blue
clPurple	Purple
clTeal	Teal
clGray	Gray
clSilver	Silver
clRed	Red
clLime	Lime green
clBlue	Blue
clFuchsia	Fuchsia
clAqua	Aqua
clWhite	White
clBackground	Current color of your Windows background
clActiveCaption	Current color of the title bar of the active window
clInactiveCaption	Current color of the title bar of inactive windows
clMenu	Current background color of menus
clWindow	Current background color of windows
clWindowFrame	Current color of window frames
clMenuText	Current color of text on menus
clWindowText	Current color of text in windows
clCaptionText	Current color of the text on the title bar of the active window
clActiveBorder	Current border color of the active window
clInactiveBorder	Current border color of inactive windows
clAppWorkSpace	Current color of the application workspace
clHighlight	Current background color of selected text
clHightlightText	Current color of selected text
clBtnFace	Current color of a button face
clBtnShadow	Current color of a shadow cast by a button
clGrayText	Current color of text that is dimmed
clBtnText	Current color of text on a button
clInactiveCaptionText	Current color of the text on the title bar of an inactive window
clBtnHighlight	Current color of the highlighting on a button

The second half of the colors listed here are Windows system colors. The color that appears depends on the color scheme users are using for Windows. Users can change these colors using the Control Panel in Program Manager. The actual color that appears will vary from system to system. For example, the color

fuchsia may appear more blue on one system than another.

When you use the Color dialog box to select a color, you are assigning a new color value to the dialog box's Color property. You can then use the value within the Color property, and assign it to the Color property of another control.

Create Method

NOTE: For more information regarding Create methods please refer to the Delphi help.

Declaration
constructor Create;

Description
The **Create** method constructs a new object instance. Create returns an instance of the type being created, allocated on the global heap. As with all constructors, Create calls the NewInstance method to allocate the memory for the instance, and the InitInstance method to initialize the allocated memory before executing its own code.

By default, Create allocates the number of bytes returned by the InstanceSize method, and initializes the allocated memory to zeros.

When declaring new component types, always add the **override** directive if your new component declares a Create method. The Create method of TComponent is virtual, so to ensure that Delphi calls the correct constructor when a user drops your component on a form, you must override the Create method.

Note:
When you override the Create constructor in a descendant object type, you should call the inherited Create to complete the initialization of inherited fields and properties. Always use the inherited keyword when calling the inherited Create, rather than specifying the ancestor type, as calling AncestorType.Create actually constructs an additional instance of that ancestor type.

Cursor Property

Example

Declaration

property Cursor: TCursor;

Description

The **Cursor** property is the image used when the mouse passes into the region covered by the control. These are the possible images:

Value	Image
crDefault	
crArrow	
crCross	
crIBeam	
crSize	
crSizeNESW	
crSizeNS	
crSizeNWSE	
crSizeWE	
crUpArrow	
crHourglass	
crDrag	
crNoDrop	
crHSplit	
crVSplit	

Destroy Method

NOTE: For more information regarding Destroy methods please refer to the Delphi help.

Declaration
destructor Destroy;

Description
The **Destroy** method destroys the object, component, or control and releases the memory allocated to it.

You seldom need to call Destroy. Objects designed with Delphi create and destroy themselves as needed, so you don't have to worry about it. If you construct an object by calling the **Create** method, you should call **Free** to release memory and dispose of the object.

Direction Property

Example

Declaration

property Direction : TDirection

Description

The **Direction** property determines which direction the progress bar will paint.

Enabled Property

Example

NOTE: This information is provided for reference only. For more information refer to the Delphi help.

Description

The **Enabled** property determines if the control responds to mouse, keyboard, or timer events, or if the data-aware controls update each time the dataset they are connected to changes.

Example for AddValue Method

Example

This example used the **AddValue** method to demonstrate incremental increases to the progress bar. The progress bar control named ProgBar1, will be changed during the creation of TForm1:

To see the **AddValue** method work **during design time** try the following:
NOTE: The AddValue method can not be seen during design time.

To see the **AddValue** method work **during program execution** try the following:

1) Save all of your current work

2) Create a new project

2a) Select New Project from the Delphi File menu

3) Place the TProgBar on the form

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) Create the actual code for the AddValue incremental change

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

```
ProgBar1.AddValue(50);
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.AddValue(50);
end;
end.
```

5) Try the application and see the results

5a) Press F9 and watch your new form. Your progress bar should display 50%

Example for Align property

Example

This example moves a progress bar control named ProgBar1, to the bottom of the form and resizes it to fill the width of the form, during the creation of TForm1:

To see the **Align** property work **during design time** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Change the Align property to see the actual alignment change**

4a) Click on Object Inspector window

4b) Select the drop down list for the **Align** property

4c) Select alBottom from the list

4d) ProgBar1 should be at the bottom of the form expanded to the width of the form

To see the **Align** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the alignment change**

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

```
ProgBar1.Align := alBottom
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.Align := alBottom
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

Example for BackColor Property

Example

This example changes the backcolor of a progress bar control named ProgBar1, to yellow during the creation of Form:

To see the **BackColor** property work **during design time** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Change the BackColor property to see the actual color change**

4a) Click on Object Inspector window

4b) Select the drop down list for the **BackColor** property

4c) Select clYellow from the list

4d) ProgBar1 should have a yellow background

To see the **BackColor** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the color change**

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

```
ProgBar1.BackColor := clYellow
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.BackColor := clYellow;
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

Example for BarShape Property

Example

This example will change the shape of the progress bar control named ProgBar1, during the creation of TForm1. This example will change the shape to a trapezoid:

To see the **BarShape** property work **during design time** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Change the BarShape property to see the actual shape change**

4a) Click on Object Inspector window

4b) Select the drop down list for the **BarShape** property

4c) Select Trapezoidal from the list

4d) ProgBar1 should be changed to a Trapezoid

To see the **BarShape** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the shape change**

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

ProgBar1.BarShape := Trapezoidal

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.BarShape := Trapezoidal;
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

Example for BarType Property

Example

This example changes the **BarType** for a progress bar control named ProgBar1, during the creation of TForm1. We will change the type to a LED style:

To see the **BarType** property work **during design time** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Change the BarType property to see the actual style change**

4a) Click on Object Inspector window

4b) Select the drop down list for the **BarType** property

4c) Select LED from the list

4d) ProgBar1 should be display black LED segments for the width of the bar

To see the **BarType** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the bar type change**

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

```
ProgBar1.BarType := LED
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.BarType := LED;
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

Example for BevelType Property

Example

This example changes the **BevelType** for a progress bar control named ProgBar1, during the creation of TForm1. The example will change the **BevelType** from **Lowered** to **Raised**:

To see the **BevelType** property work **during design time** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Change the BevelType property to see the bevel change**

4a) Click on Object Inspector window

4b) Select the drop down list for the **BevelType** property

4c) Select Raised from the list

4d) ProgBar1 should have the bevel raised

To see the **BevelType** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the bevel change**

4a) **Double click on TForm1 in a blank area**

4b) **Add GrphFunc to the uses line right before ProgBar.**

Your uses statement should look like the following

```
uses
    SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls, Forms,
Dialogs, GrphFunc, ProgBar;
```

4c) **Type in the following line of code in the edit window after the begin.**

```
ProgBar1.BevelType := Raised
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.BevelType := Raised;
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

Example for BevelWidth Property

Example

This example changes the bevel width on a progress bar control named ProgBar1, during the creation of TForm1. The example will change the width from 2 to 10:

To see the **BevelWidth** property work **during design time** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Change the BevelWidth property to see the actual width change**

4a) Click on Object Inspector window

4b) Click on the **BevelWidth** edit field

4c) Enter 10 and press enter

4d) ProgBar1 should have a large bevel

To see the **BevelWidth** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the width change**

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

```
ProgBar1.BevelWidth := 10;
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.BevelWidth := 10;
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

Example for Bitmap Property

Example

This example assigns a bitmap for the foreground of a progress bar control named ProgBar1, during the creation of TForm1. This example will use the arcade.bmp found in your windows directory:

To see the **Bitmap** property work **during design time** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Change the Bitmap property to see the actual bitmap change**

4a) Click on Object Inspector window

4b) Double click the **Bitmap** property

4c) Click Load from the Picture Editor dialog

4d) In the Load Picture dialog navigate to your MS-Windows subdirectory

4e) Double click the arcade.bmp found in the file list

4f) Click OK on the Picture Editor dialog

4g) Click the edit field for Value property

4h) Enter the number 75 for the Value property

4i) ProgBar1 should have the arcade.bmp 3/4 way across the progress bar

To see the **Bitmap** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the bitmap change**

4a) Double click on TForm1 in a blank area

4b) Type in the following lines of code in the edit window after the begin.

NOTE: Substitute your MS-Windows path if it is different than C:\WINDOWS

```
ProgBar1.Bitmap.LoadFromFile('C:\windows\arcade.bmp');
ProgBar1.Value := 75;
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.Bitmap.LoadFromFile('C:\windows\arcade.bmp');
        ProgBar1.Value := 75;
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

If you receive an error check your pathname of the bitmap file.

Example for BorderType Property

Example

This example changes the border type for a progress bar control named ProgBar1, during the creation of TForm1. This example will change the border type from **bsNone** to **bsSingle**:

To see the **BorderType** property work **during design time** try the following:

1) Save all of your current work

2) Create a new project

2a) Select New Project from the Delphi File menu

3) Place the TProgBar on the form

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) Change the BorderType property to see the actual border change

4a) Click on Object Inspector window

4b) Select the drop down list for the **BorderType** property

4c) Select bsSingle from the list

4d) ProgBar1 should have a thin black border sorrounding it

NOTE: If you are using a high-resolution monitor you may want to click on the blank form once to remove the focus from ProgBar1, to better see the border.

To see the **BorderType** property work **during program execution** try the following:

1) Save all of your current work

2) Create a new project

2a) Select New Project from the Delphi File menu

3) Place the TProgBar on the form

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) Create the actual code for the border change

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

ProgBar1.BorderType := bsSingle

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.BorderType := bsSingle;
end;
end.
```

5) Try the application and see the results

5a) Press F9 and watch your new form

Example for Cursor Property

Example

This example changes the cursor for a progress bar control named ProgBar1, during the creation of TForm1. This example changes the cursor from the **csDefault** to **crlBeam**:

NOTE: The Cursor property can be changed during design, but you will not see the results until runtime.

To see the **Cursor** property work **during program execution** try the following:

1) Save all of your current work

2) Create a new project

2a) Select New Project from the Delphi File menu

3) Place the TProgBar on the form

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) Create the actual code for the cursor change

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

ProgBar1.Cursor := criBeam;

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.Cursor := criBeam;
end;
end.
```

5) Try the application and see the results

5a) Press F9 and watch your new form

Pass the mouse over the progress bar and watch the cursor change to the IBEAM. If you receive an "unknown identifier" compile message, you probably misspelled criBeam. Check the third letter and make sure it is an i.

Example for Direction Method

Example

This example changes the paint direction of a progress bar control named ProgBar1, during the creation of TForm1. This example changes the direction from **RightDirect** to **DownDirect**:

To see the **Direction** property work **during design time** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Change the Direction property to see the actual direction change**

4a) Click on Object Inspector window

4b) Select the drop down list for the **Direction** property

4c) Select DownDirect from the list

4d) Click on the Value property to edit the value

4e) Enter 50 for Value property and press enter

4f) ProgBar1 should have the top half of the progress bar filled in

To see the **Direction** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the direction change**

NOTE: Add GrphFunc to the uses statement right before ProgBar. Your uses statement should look like the following:

```
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, GrphFunc, ProgBar;
```

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

```
ProgBar1.Direction := DownDirect;
ProgBar1.Value := 50;
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.Direction := DownDirect;
        ProgBar1.Value := 50;
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

Example for Enabled Property

Example

This example disables a progress bar control named ProgBar1, during the creation of TForm1:

To see the **Enabled** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the enabled change**

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

```
ProgBar1.Enabled := FALSE
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.Enabled := FALSE
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

NOTE: You won't see much with this example. The control is disabled and there are no visible signs indications.

Example for Font Property

Example

This example changes the font on a progress bar control named ProgBar1, during the creation of TForm1. This example will change the font from system to arial:

To see the **Font** property work **during design time** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Change the Direction property to see the actual direction change**

4a) Click on Object Inspector window

4b) Expand the **Font** property

4c) Select Arial from the Name drop down list

4d) Click on the Value property to edit the value

4e) Enter 50 for Value property and press enter

4f) ProgBar1 should display 50% with the Arial font

To see the **Font** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the font change**

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

```
ProgBar1.Font.Name := 'Arial';
ProgBar1.Value := 50;
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.Font.Name := 'Arial';
        ProgBar1.Value := 50;
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

Example for ForeColor Property

Example

This example changes the forecolor on a progress bar control named ProgBar1, during the creation of TForm1. This example will change the forecolor from **clBtnFace** to **clYellow**:

To see the **Font** property work **during design time** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Change the ForeColor property to see the actual color change**

4a) Click on Object Inspector window

4b) Select clYellow from the Forecolor drop down list

4c) Enter 50 for Value property and press enter

4d) ProgBar1 should display display 50% of the bar in yellow

To see the **ForeColor** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the color change**

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

```
ProgBar1.ForeColor := clYellow;
ProgBar1.Value := 50;
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.ForeColor := clYellow;
        ProgBar1.Value := 50;
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

Example for Hint Property

Example

This example changes the hint line for a progress bar control named ProgBar1, during the creation of TForm1:

To see the **Hint** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the hint change**

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

```
ProgBar1.Hint := "A Nice Hint";
ProgBar1.ShowHint := TRUE;
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.Hint := 'A Nice Hint';
        ProgBar1.ShowHint := TRUE;
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

Move the mouse over the progress bar and in a couple of seconds the hint box will show.

6) **Now for a little fun (NOTE: You will need to have the VisualHelpCloud component installed for this to work)**

6a) Click on the VisualPROS-1 tab found in VCL tabs

6b) Select the HelpCloud icon and double click

6c) Press F9 and watch your new form

Move the mouse over the progress bar, and in a couple of seconds the help cloud will show your help. See how easy it is to use VisualPROS to add dramatic impact to your applications!

Example for Left Property

Example

This example moves a progress bar control named ProgBar1, during the creation of TForm1. This example uses the left property to move the progress bar:

To see the **Left** property work **during design time** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Change the Left property to see the actual movement**

4a) Click on Object Inspector window

4b) Click on the **Left** property edit field

4c) Enter 200 and press enter

4d) ProgBar1 should have moved left

To see the **Left** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the Left change**

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

```
ProgBar1.Left := 200;
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.Left := 200;
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

Example for MaxValue Property

Example

This example changes the maxvalue on a progress bar control named ProgBar1, during the creation of TForm1. This example will change the maxvalue from 100 to 200:

To see the **MaxValue** property work **during design time** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Change the MaxValue property to see the actual value change**

4a) Click on Object Inspector window

4b) Double click on the **Value** property

4c) Enter 100 and press Enter

The progress bar should be full at 100%

4d) Double click on the **MaxValue** property

4e) Enter 200 for the MaxValue property and press enter

4f) ProgBar1 should display 50%

To see the **MaxValue** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the value change**

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

```
ProgBar1.Value := 100;
ProgBar1.MaxValue := 200;
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.Value := 100;
        ProgBar1.MaxValue := 200;
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

Example for MinValue Property

Example

This example changes the minvalue on a progress bar control named ProgBar1, during the creation of TForm1. This example will change the minvalue from 0 to 10:

To see the **MinValue** property work **during design time** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Change the MinValue property to see the actual value change**

4a) Click on Object Inspector window

4b) Double click on the **Value** property

4c) Enter 25 and press Enter

The progress bar should be full at 25%

4d) Double click on the **MinValue** property

4e) Enter 10 for the MinValue property and press enter

4f) ProgBar1 should display 16%

NOTE: MaxValue must equal 100 for this to work.

To see the **MinValue** property work **during program execution** try the following:

1) **Save all of your current work**

2) Create a new project

2a) Select New Project from the Delphi File menu

3) **Place the TProgBar on the form**

3a) Click on the VisualPROS-1 tab found in VCL tabs

3b) Select the ProgBar icon and double click

4) **Create the actual code for the value change**

4a) Double click on TForm1 in a blank area

4b) Type in the following line of code in the edit window after the begin.

```
ProgBar1.Value := 25;
ProgBar1.MinValue := 10;
```

Your code should look like the following

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        ProgBar1.Value := 25;
        ProgBar1.MinValue := 10;
end;
end.
```

5) **Try the application and see the results**

5a) Press F9 and watch your new form

Font Property

See Also Example

Declaration

property Font: TFont;

Description

The **Font** property is a font object that controls the attributes of text written on or in the component, or object, or sent to the printer. To modify a font, you change the value of the Color, Name, Size or Style properties of the font object.

ForeColor Property

See Also Example

Declaration

property ForeColor: TColor;

Description

The **ForeColor** property is used to set the foreground color of the progress bar. Even when using a bitmap or LED for the foreground, the selected backcolor is still used. The ForeColor property uses the same values as the Color property.

Height Property

Declaration

property Height: **Integer**;

Description

The **Height** property determines vertical size of a component or object.

Hint Property

Example

Declaration

property Hint: **string**;

Description

The **Hint** property is the text string that can appear when the OnHint event occurs, which happens when the user moves the mouse pointer over a control or menu item. The code within the OnHint event handler determines how the string is displayed. A common use of an OnHint event handler is to display the hint as the caption of a panel component, that is being used as a status bar.

You can have a Help Hint, a box containing help text, appear for a control when the user moves the mouse pointer over the control, and pauses momentarily. This is how:

1 Specify a Hint value for each control you want a Help Hint to appear for.
2 Set the ShowHint property of each control to True.
3 At run time, set the value of application's ShowHint property to True.

If the application's ShowHint property is False, the Help Hint won't appear.

If a control has no Hint value specified, but its parent control does, the control uses the Hint value of the parent control, as long as the control's ShowHint property is True.

With SHORELINE's **VisualHelpCloud** you can create floating bubble with your applications.

Key Property

The most significant properties: Generally having an effect on the behavior of the component. You probably want to learn these properties first.

Left Property

Example

Declaration

property Left: **Integer**;

Description

The **Left** property determines the horizontal coordinate of the left edge of a component, relative to the form in pixels. For forms, the value of the Left property is relative to the screen in pixels. The default value is -1.

MaxValue Property

See Also Example

Declaration

property MaxValue : **Longint;**

Description

The **MaxValue** property establishes the upper limit for the progress bar.

Methods

Declaration
procedure AddValue (number: **Longint**);
property Percentage: **Longint**;

MinValue Property

See Also Example

Declaration

property MinValue : **Longint;**

Description

The **MinValue** property establishes the lower limit for the progress bar.

Name Property

Declaration

property Name: TProgBar;

Description

The **Name** property contains the name of the component as referenced by other components. By default, Delphi assigns sequential names based on the type of the component, such as 'ProgBar1', 'ProgBar2' and so on. You may change these to suit your needs.

Note: Change component names only at design time.

NumSegments Property

See Also

Declaration

property NumSegments : **Byte;**

Description

NumSegments defines how many segments are available for the LED display option.

OnClick Event

Description

An **OnClick** event occurs when the user clicks the component

OnDblClick Event

Description

The **OnDblClick** event occurs when the user double-clicks the mouse button, while the mouse pointer is over the component.

OnDragDrop Event

Description

The **OnDragDrop** event occurs when the user drops an object being dragged. Use the OnDragDrop event handler to specify what you want to happen when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

OnDragOver Event

Description

The **OnDragOver** event occurs when the user drags an object over a component. Usually you'll use an OnDragOver event to accept an object, so the user can drop it.

The OnDragOver event accepts an object when its Accept parameter is True.

Usually, you will want the cursor to change shape, indicating that the control can accept the dragged object if the user drops it. You can change the shape of the cursor by changing the value of the **DragCursor** property for the control, at either design or run time, before an OnDragOver event occurs.

OnEndDrag Event

Description

The **OnEndDrag** event occurs whenever the dragging of an object ends, either by dropping the object, or by canceling the dragging. Use the OnEndDrag event handler to specify any special processing you want to occur when the dragging stops. If the dragged object was dropped and accepted by the control, then the Target parameter of the OnEndDrag event is True. If the object was not dropped successfully, the value of Target is **nil**.

OnEnter Event

Description

The **OnEnter** event occurs when a component becomes active. Use the OnEnter event handler to specify any special processing you want to occur when a component becomes active.

Note:

The OnEnter event does not occur when switching between forms, or between another Windows application and your application.

The OnEnter event for a TPanel or THeader component never occurs, as panels and headers never receive focus.

OnExit Event

Description

The **OnExit** event occurs when the input focus shifts away from one control to another. Use the OnExit event handler when you want special processing to occur when this control ceases to be active.

Note:

The OnExit event does not occur when switching between forms, or between another Windows application and your application.

The OnExit event for a TPanel or THeader component never occurs, as panels and headers never receive focus.

The ActiveControl property is updated before an OnExit event occurs.

OnMouseDown Event

Description

The **OnMouseDown** event occurs when the user presses a mouse button with the mouse pointer over a control. Use the OnMouseDown event handler when you want some processing to occur, as a result of pressing a mouse button.

The Button parameter of the OnMouseDown event identifies which mouse button was pressed. By using the Shift parameter of the OnMouseDown event handler, you can respond to the state of the mouse buttons and shift keys. Shift keys are the Shift, Ctrl and Alt keys.

OnMouseMove Event

Description

The **OnMouseMove** occurs when the user moves the mouse pointer while the mouse pointer is over a control. Use the OnMouseMove event handler when you want something to happen while the mouse pointer moves within the control.

By using the Shift parameter of the OnMouseDown event handler, you can respond to the state of the mouse buttons and shift keys. Shift keys are the Shift, Ctrl and Alt keys.

OnMouseUp Event

Description

The **OnMouseUp** event occurs when the user releases a mouse button that was pressed with the mouse pointer over a component. Use the OnMouseUp event handler when you want processing to occur when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses, and shift key plus mouse button combinations. Shift keys are the Shift, Ctrl and Alt keys.

OnResize Event

Description

The **OnResize** event occurs whenever the form is resized while an application is running. Use the OnResize event handler when you want something to happen in your application when the form is resized.

Overriding Methods

NOTE: For more information regarding overriding methods please refer to the Delphi help.

Overriding a method means extending or refining it, rather than replacing it. That is, a descendant object type can redeclare and reimplement any of the methods declared in its ancestors. One cannot override static methods, because declaring a static method with the same name as an inherited static method replaces the inherited method completely.

To override a method in a descendant object type, add the directive **override** to the end of the method declaration.

Using **override** will cause a compile-time error if:

The method does not exist in the ancestor object

The ancestor's method of that name is static

The declarations are not otherwise identical (names and types of parameters, procedure vs. function, and so on)

ParentColor Property

See Also

Declaration

property ParentColor: **Boolean**;

Description

The **ParentColor** property determines where a control looks for its color information. If ParentColor is True, then the control uses the color in its parent component's Color property. If ParentColor is False, the control uses its own Color property. Except for the radio group, database radio group, label and database text controls, the default value is False.

By using ParentColor, you can ensure that all the controls on a form have a uniform appearance. For example, if you change the background color of your form to gray, by default, the controls on the form will also have a gray background.

To specify a different color for a particular control, specify the desired color as the value of that control's Color property, and ParentColor becomes False automatically.

ParentFont Property

See Also

Declaration

property ParentFont: **Boolean**;

Description

The **ParentFont** property determines where a control looks for its font information. If ParentFont is True, then the control uses the font in its parent component's Font property. If ParentFont is False, the control uses its own Font property.

By using ParentFont, you can ensure that all the controls on a form have a uniform appearance. For example, if you want all the controls in a form to use 12-point Courier for their font, you can set the form's Font property to that font. By default, all the controls on that form will use the same font.

To specify a different font for a particular control, specify the desired font as the value of the control's Font property, and ParentFont becomes False automatically.

When the ParentFont is True for a form, the form uses the value of the application's Font property.

Percentage Property

Declaration

property Percentage: **Longint**;

Description

The **Percentage** property is a Read Only property and only available at Runtime. The property returns the current value of the progress bar.

Read Only

Properties with this symbol are read only and cannot be changed. Usually provided so that application code can inspect certain characteristics of a component.

Runtime Only

Properties with this symbol are only available at runtime. In design mode these properties will not be shown in the object inspector. Most runtime only properties are read only as well.

See Also

About
Align
BackColor
BarShape
BarType
BevelType
BevelWidth
Bitmap
BorderType
BrushStyle
Color
Cursor
Direction
Font
ForeColor
Height
Hint
Left
MaxValue
MinValue
Name
NumSegments
ParentColor
Seg1Color ... Seg3Color
Seg1Count ... Seg3Count
SegOffColor
SegSpacing
ShowText
Tag
Top
TrapezoidDir
TrapezoidMin
TrapezoidShape
Value
Width

See Also

Color
Font Color
ForeColor
ParentColor
Seg1Color ... Seg3Color
SegOffColor

See Also

BarType
Direction
TrapezoidDir
TrapezoidMin
TrapezoidShape

See Also

BarShape
BevelType
BevelWidth
BorderType
NumSegments
Seg1Color ... Seg3Color
Seg1Count ... Seg3Color
SegOffColor
SegSpace

See Also

BevelType
BevelWidth

See Also

BackColor
BrushStyle
Color
ForeColor
ParentColor
Seg1Color ... Seg3Color
SegOffColor

See Also

NumSegments
Seg1Color ... Seg3Color
Seg1Count ... Seg3Count
SegOffColor
SegSpacing

See Also

BarShape
BarType
TrapezoidDir
TrapezoidMin
TrapezoidShape

See Also

BarShape
BarType
Bitmap
BorderType
ShowText

See Also

BackColor
Font
ForeColor
Height
ShowText

See Also

MaxValue
MinValue
TrapezoidMin

Seg1Color Property

See Also

Declarations

property Seg1Color : TColor;

Description

Seg1Color property is used to define the color for the first segment group when using the LED option for the progress bar.

Seg1Count Property

See Also

Declaration

property Seg1Count : **Byte;**

Description

Seq1Count property determines how many segments are dedicated to the first segment.

Seg2Color Property

See Also

Declarations

property Seg2Color : TColor;

Description

Seg2Color property is used to define the color for the second segment group when using the LED option for the progress bar.

Seg2Count Property

See Also

Declaration

property Seg2Count : **Byte;**

Description

Seq2Count property determines how many segments are dedicated to the second segment.

Seg3Color Property

See Also

Declarations

property Seg3Color : TColor;

Description

Seg3Color property is used to define the color for the third segment group when using the LED option for the progress bar.

Seg3Count Property

See Also

Declaration

property Seg3Count : **Byte;**

Description

Seq3Count property determines how many segments are dedicated to the third segment.

SegOffColor Property

See Also

Declarations

property SegOffColor : TColor;

Description

SegOffColor property is used to define the forecolor for the segment group when the is not illuminated.

SegSpacing Property

See Also

Declaration

property SegSpacing : **Byte;**

Description

The **SegSpacing** property determines the pixel spacing between each segment.

ShowHint Property

Declaration
property ShowHint: **Boolean**;

Description
The **ShowHint** property determines if the control should display a Help Hint when the user's mouse pointer momentarily rests on the control. The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. If ShowHint property is True, the Help Hint will appear.

If ShowHint is False, the Help Hint may or may not appear. If ParentShowHint is also False, the Help Hint won't appear. If, however, ParentShowHint is True, whether or not the Help Hint appears depends on the setting of the ShowHint property of the control's parent. For example, imagine a check box within a group box. If the ShowHint property of the group box is True and the ParentShowHint property of the check box is True, but the ShowHint property of the check box is False, the check box will still display its Help Hint.

The default value is False.

Changing the ShowHint value to True automatically sets the ParentShowHint property to False.

ShowText Property

Declaration

property ShowText : **Boolean;**

Description

The **ShowText** property allows you to disable the percent text from being displayed. In some cases where you just need the LED display, and not the percentage shown, this property should be set to FALSE.

Style Property

NOTE: The Style property is not directly available to TProgBar it is here for reference purposes only.

Declaration
property Style: TBevelStyle;

Description
The value of the **Style** property determines if the bevel is raised or lowered. These are the possible values:

Value	Meaning
bsLowered	The bevel is lowered.
bsRaised	The bevel is raised.

TBevelStyle Type

Declaration

```
type
  TBevelStyle = (bsLowered, bsRaised);
```

Description

The **TBevelStyle** type defines the possible values of the Style property, of the TBevel component.

TBitmap Type

NOTE: More information is available in the standard Delphi on-line help.

Description

A **TBitmap** object contains a bitmap graphic (.BMP file format). A TBitmap encapsulates a Windows HBITMAP and an HPALETTE, and manages the realizing of the palette automatically.

The canvas of the TBitmap is a TCanvas object specified by the Canvas property. The palette of the TBitmap is specified by the Palette property.

The height and width in pixels of the bitmap are specified by the Height and Width properties, respectively.

If the Monochrome property is set to False, then the bitmap is displayed in color. If Monochrome is set to True,
then the bitmap is displayed in monochrome.

To load a bitmap from a file, call the LoadFromFile method. To save a bitmap to a file, call SaveToFile.

To draw a bitmap on a canvas, call the Draw or StretchDraw methods of a TCanvas object, passing a TBitmap as a parameter.

When the bitmap is modified, an OnChange event occurs.

In addition to these properties, methods, and events, this object also has the methods that apply to all objects.

TBorderStyle Type

Declaration

type

```
  TBorderStyle = bsNone..bsSingle;
```

Description

TBorderStyle is the type of BorderStyle property for controls. The BorderStyle property for forms and windows uses the type TFormBorderStyle.

TBrush Property

NOTE: This information is provided for reference purposes only. Please refer to the standard Delphi help for more information.

Description

A **TBrush** object is used when filling solid shapes, such as rectangles and ellipses. The interior of the shape is filled with a color or pattern. TBrush encapsulates a Windows HBRUSH.

The color of the brush is specified by the Color property, and the pattern is specified by the Style property. If a bitmap is specified by the Bitmap property, the pattern of the brush is defined by the bitmap, rather than the Style property.

If the brush is modified, an OnChange event occurs.

In addition to these properties, methods, and events, this object also has the methods that apply to all objects.

TBrushStyle Type

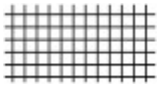
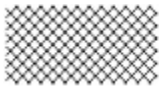
Declaration

type

TBrushStyle = (bsSolid, bsClear, bsHorizontal, bsVertical, bsFDiagonal, bsBDiagonal, bsCross, DiagCross);

Description

The **TBrushStyle** type is used by the Style property to determine the pattern of a TBrush object.

Hatch	Pattern
bsSolid	
bsClear	
bsBDiagonal	
bsFDiagonal	
bsCross	
bsDiagCross	
bsHorizontal	
bsVertical	

TColor Type

Declaration

type

```
  TColor = -(COLOR_ENDCOLORS + 1)..$02FFFFFF;
```

Description

The **TColor** type is used to specify the color of an object.

The Graphics unit contains definitions of useful constants for TColor. These constants map either directly to the closest matching color in the system palette (for example, clBlue maps to blue), or to the corresponding system screen element color, defined in the Color section of the Windows Control panel (for example, clBtnFace maps to the system color for button faces).
The constants that map to the closest matching system colors are: clAqua, clBlack, clBlue, clDkGray, clFuchsia, clGray, clGreen, clLime, clLtGray, clMaroon, clNavy, clOlive, clPurple, clRed, clSilver, clTeal, clWhite and clYellow.

The constants that map to the system screen element colors are: clActiveBorder, clActiveCaption, clAppWorkSpace, clBackground, clBtnFace, clBtnHighlight, clBtnShadow, clBtnText, clCaptionText, clGrayText, clHighlight, clHighlightText, clInactiveBorder, clInactiveCaption, clInactiveCaptionText, clMenu, clMenuText, clScrollBar, clWindow, clWindowFrame and clWindowText.

If you specify TColor as a specific 4-byte hexadecimal number instead of using the constants defined in the graphics unit, the low three bytes represent RGB color intensities for blue, green, and red, respectively. The value $00FF0000 represents full-intensity, pure blue, $0000FF00 is pure green, and $000000FF is pure red. $00000000 is black and $00FFFFFF is white.

If the highest-order byte is zero ($00), then the color obtained is the closest matching color in the system palette. If the highest-order byte is one ($01), the color obtained is the closest matching color in the currently realized palette. If the highest-order byte is two ($02), the value is matched with the nearest color in the logical palette of the current device context.

To work with logical palettes, you must select the palette with the Windows API function SelectPalette. To realize a palette, you must use the Windows API function RealizePalette.

TCursor Type

Declaration

```
type
  TCursor = -32768..32767;
```

Description

The **TCursor** type defines the different kinds of standard cursors a component can have. TCursor is the type of the Cursor property, and the DragCursor property.

TDirection Type

Declaration

type

```
    TDirection      = ( UpDirect, DownDirect, LeftDirect, RightDirect );
```

TFont Object

NOTE: This information provided for reference only. For more information refer to the Delphi help.

Description

A **TFont** object defines the appearance of text. TFont encapsulates a Windows HFONT.

A TFont object defines a set of characters by specifying their height, font family (typeface), name and so on. The height is specified by the Height property; The typeface is specified by the Name property; The size in points is specified by the Size property; The color is specified by the Color property; The attributes of the font (bold, italic, and so on) are specified by the Style property.

When a font is modified, an OnChange event occurs.

TPBarShape Type

type

```
TPBarShape = ( Rectangular, Trapezoidal );
```

TPBarType Type

type

TPBarType = (Normal, Bar3D, LED);

TProgBar Component

Properties Methods

SHORELINE SOFTWARE **35-31 Talcottville Rd. #123, Vernon, CT 06066-4030**
Technical Support: **Phone: (203) 870-5707 24-Hour Fax: (203) 870-5727**
CompuServe 70541,2436

Description
Use a TProgBar to provide a graphical status to the user. ProgBars can be solid fill, or led with multi-color segments. A bitmap may be used for the progress bar as well. You can also refer to the Progress Bar as a Percent Bar.

Feature List

- Standard percent bars
- Flicker free performance
- LED segment display
- Complete font control
- Progress Bar can be a bitmap
- 3D options
- Easy setup and use

Key Properties
Use the MinValue and MaxValue properties to set the lower and upper range for your percent bar. Use the Value property to set the current position within that range. If the value property is set beyond the min or max value, it will be set to the min or max value.

Use the BarType to select a 3D bar, LED bar or Normal bar type. If you want to change the shape of the bar use BarShape to select Rectangular or Trapezoidal.

If you want to use a bitmap for the progress bar, double click the Bitmap property to load the Picture Editor. The Picture Editor is included with Delphi.

When using the LED bar you have several properties which you can adjust. These properties include: NumSegments, SegOffColor, SegSpacing and three independent segments, colors and counts. (Seg1Color, Seg1Count)

Use the ShowText property to enable, or disable, percent text displayed on the progress bar. The percent text is not available when the LED bar is used.

About SHORELINE SOFTWARE

Properties

About
Align
BackColor
BarShape
BarType
BevelType
BevelWidth
Bitmap
BorderType
BrushStyle
Color
Cursor
Direction
Font
ForeColor
Height
Hint
Left
MaxValue
MinValue
Name
NumSegments
ParentColor
ParentFont
Percentage
Seg1Color
Seg1Count
Seg2Color
Seg2Count
Seg3Color
Seg3Count
SegOffColor
SegSpacing
ShowText
Tag
Top
TrapezoidDir
TrapezoidMin
TrapezoidShape
Value
Width

TTrapDirect Type

See Also

Declaration

type

```
  TTrapDirect     = ( LargeToSmall, SmallToLarge );
```

TTrapShape Type

See Also

Declaration

```
type
  TPBarShape = ( TSLeft, TSCenter, TSRight );
```

Tag Property

Declaration

property Tag: **Longint**;

Description

The **Tag** property is available to store an integer value as part of a component. While the Tag property has no meaning to Delphi, your application can use the property to store a value for its special needs.

Test

asdfasdf

Top Property

Declaration

property Top: **Integer**;

Description

The **Top** property determines the y coordinate of the top left corner of a control, relative to the form in pixels. For forms, the value of the Top property is relative to the screen in pixels.

TrapezoidDir Property

See Also

Declaration

property TrapezoidDir : TTrapDirect;

Description

The **TrapezoidDir** property identifies which direction you wish the trapezoid to point. The property is only available if the BarShape has been set to **Trapezoidal**.

TrapezoidMin Property

See Also

NOTE THIS NEEDS TO BE CHANGED!!!

Declaration

property TrapezoidMin : **Integer**;

Description

The **TrapezoidMin** property determines the . The BarShape must be set to **Trapezoidal**.

TrapezoidShape Property

See Also

Declaration

property TrapezoidShape: TTrapShape

Description

The **TrapezoidShape** property determines the visual characteristics of the shape. The BarShape must be set to **Trapezoidal**.

Value Property

Declaration
property Value: **Longint;**

Description
The **Value** property contains the initialized value for the progress bar. This is where the progress will actually start within the MinValue and MaxValue range.

Visible Property

Declaration
property Visible: **Boolean**;

Description
The **Visible** property determines whether or not the component appears onscreen. If Visible is True, the component appears. If Visible is False, then the component is not visible.

For controls, calling the Show method makes the control's Visible property True, but it also performs other actions to ensure that the user can view the control.

For field components, the Visible property determines if a field can be displayed in a TDBGrid component. If Visible is False, the field is not displayed.

The default value is True for all components, except for forms.

Where to send your resume

Please send your resume and other CV related materials to:

SHORELINE SOFTWARE
35-31 Talcottville Road, #123
Vernon, CT 06066-4030

ATTN: Glenn A. Field

Phone: 800-261-9198
Fax: 203-870-5727

We will contact you after receiving your resume.

Width Property

Declaration

property Width: **Integer**;

Description

The **Width** property determines horizontal size.