

in delphi code create a shape component that contains a label where text can be displayed. It is to create the text bubbles in a text messaging app

To create a custom component in Delphi that combines a shape and a label to mimic the appearance of a text bubble in a messaging app, you'd typically inherit from an existing component (like `TPanel` or `TCustomControl`) and then override its painting methods and add the desired functionality.

Here's a basic example:

- 1. Start with a new package:**
 - Open Delphi.
 - Click `File -> New -> Package`.
 - Save the package as `BubblePackage.dpk`.
- 2. Add a new component:**
 - In the package, click `File -> New -> Component`.
 - Set the ancestor type as `TCustomControl`.
 - Name it `TBubbleText`.
 - Save it as `BubbleText.pas`.
- 3. Code for the `TBubbleText` component:**

```
delphi
unit BubbleText;

interface

uses
  System.Classes, Vcl.Controls, Vcl.StdCtrls, Vcl.Graphics, System.Types,
  System.UITypes;

type
  TBubbleText = class(TCustomControl)
  private
    FLabel: TLabel;
    procedure SetText(const Value: string);
    function GetText: string;
  protected
    procedure Paint; override;
    procedure Resize; override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property Text: string read GetText write SetText;
    property Color;
    property Font;
    property Align;
    property Anchors;
    property Constraints;
    property Visible;
  end;

procedure Register;

implementation
```

```

procedure Register;
begin
  RegisterComponents('Samples', [TBubbleText]);
end;

{ TBubbleText }

constructor TBubbleText.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Width := 150;
  Height := 50;

  FLabel := TLabel.Create(Self);
  FLabel.Parent := Self;
  FLabel.AutoSize := False;
  FLabel.WordWrap := True;
  FLabel.Align := alClient;
  FLabel.Layout := tlCenter;
  FLabel.Transparent := True;
end;

destructor TBubbleText.Destroy;
begin
  FLabel.Free;
  inherited;
end;

function TBubbleText.GetText: string;
begin
  Result := FLabel.Caption;
end;

procedure TBubbleText.Paint;
var
  Rect: TRectF;
begin
  inherited;
  Canvas.Pen.Color := Color;
  Canvas.Brush.Color := Color;
  Rect := RectF(0, 0, Width, Height);

  // Drawing the rounded rectangle as the bubble shape
  Canvas.RoundRect(Rect, 15, 15);
end;

procedure TBubbleText.Resize;
begin
  inherited;
  Invalidate; // Redraw on resize
end;

procedure TBubbleText.SetText(const Value: string);
begin
  FLabel.Caption := Value;
end;

end.

```

4. Compile and Install:

- Add the component to the package by right-clicking the package in the Project Manager, selecting Add, and then choosing `BubbleText.pas`.
- Right-click the package in the Project Manager and choose `Compile`.
- Right-click the package again and choose `Install`.

After you've completed the above steps, the new `TBubbleText` component should appear in the tool palette under the "Samples" category. You can then drag and drop it onto your forms and set the `Text` property to display messages in the bubble. Adjust the appearance (colors, fonts, etc.) as required.

Can you add a property that sets whether the bubble is for what the user has typed or for the other end. Based on this reduce the size of drawn rect so that for the text that the user has typed the rect is left justified and for the other end it is right justified.

Sure! I'll add a property named `BubbleType` which can either be `btUser` or `btOther`. Depending on the value of this property, the bubble will be drawn either left justified or right justified.

Here's the modified code:

```
delphi
unit BubbleText;

interface

uses
  System.Classes, Vcl.Controls, Vcl.StdCtrls, Vcl.Graphics, System.Types,
  System.UITypes;

type
  TBubbleType = (btUser, btOther);

  TBubbleText = class(TCustomControl)
  private
    FLabel: TLabel;
    FBubbleType: TBubbleType;
    procedure SetText(const Value: string);
    function GetText: string;
    procedure SetBubbleType(const Value: TBubbleType);
  protected
    procedure Paint; override;
    procedure Resize; override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property Text: string read GetText write SetText;
    property BubbleType: TBubbleType read FBubbleType write SetBubbleType
  default btUser;
    property Color;
    property Font;
    property Align;
    property Anchors;
    property Constraints;
    property Visible;
  end;
```

```

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Samples', [TBubbleText]);
end;

{ TBubbleText }

constructor TBubbleText.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Width := 150;
  Height := 50;

  FLabel := TLabel.Create(Self);
  FLabel.Parent := Self;
  FLabel.AutoSize := False;
  FLabel.WordWrap := True;
  FLabel.Align := alClient;
  FLabel.Layout := tlCenter;
  FLabel.Transparent := True;

  FBubbleType := btUser; // Default type
end;

destructor TBubbleText.Destroy;
begin
  FLabel.Free;
  inherited;
end;

function TBubbleText.GetText: string;
begin
  Result := FLabel.Caption;
end;

procedure TBubbleText.Paint;
var
  Rect: TRectF;
begin
  inherited;

  Canvas.Pen.Color := Color;
  Canvas.Brush.Color := Color;

  case FBubbleType of
    btUser:
      begin
        Rect := RectF(0, 0, Width * 0.75, Height);
      end;
    btOther:
      begin
        Rect := RectF(Width * 0.25, 0, Width, Height);
      end;
  end;

  // Drawing the rounded rectangle as the bubble shape

```

```

    Canvas.RoundRect (Rect, 15, 15);
end;

procedure TBubbleText.Resize;
begin
    inherited;
    Invalidate; // Redraw on resize
end;

procedure TBubbleText.SetBubbleType(const Value: TBubbleType);
begin
    if FBubbleType <> Value then
    begin
        FBubbleType := Value;
        Invalidate; // Redraw when BubbleType changes
    end;
end;

procedure TBubbleText.SetText(const Value: string);
begin
    FLabel.Caption := Value;
end;

end.

```

In this version:

- I've added the `TBubbleType` enumerated type to represent the two kinds of bubbles.
- The `BubbleType` property was added to the `TBubbleText` component. This property determines how the bubble will be drawn.
- The `Paint` method has been updated to conditionally set the `Rect` value based on the `BubbleType`.

When you use this component on a form, you can set the `BubbleType` property to `btUser` or `btOther` to control the appearance of the bubble. The bubble will be left justified for `btUser` and right justified for `btOther`.

Is the text contained in the adjusted rect now?

Oops! My apologies for missing that detail. When we adjust the rect for the bubble drawing, we should also adjust the label's position and width accordingly to ensure that the text is contained within the bubble.

I'll modify the code to handle the label's position and dimensions according to the `BubbleType`:

```

delphi
procedure TBubbleText.Paint;
var
    Rect: TRectF;
begin
    inherited;

    Canvas.Pen.Color := Color;
    Canvas.Brush.Color := Color;

```

```

case FBubbleType of
  btUser:
    begin
      Rect := RectF(0, 0, Width * 0.75, Height);
      FLabel.SetBounds(5, 0, Trunc(Width * 0.75) - 10, Height);
    end;
  btOther:
    begin
      Rect := RectF(Width * 0.25, 0, Width, Height);
      FLabel.SetBounds(Trunc(Width * 0.25) + 5, 0, Trunc(Width * 0.75) -
10, Height);
    end;
end;

// Drawing the rounded rectangle as the bubble shape
Canvas.RoundRect(Rect, 15, 15);
end;

procedure TBubbleText.SetBubbleType(const Value: TBubbleType);
begin
  if FBubbleType <> Value then
    begin
      FBubbleType := Value;
      Invalidate; // Redraw when BubbleType changes
    end;
end;

```

Now, within the adjusted `Rect`, I've set the bounds of the `FLabel` to make sure the text it displays is contained within the bubble's rectangle. The values 5 and -10 are for padding inside the bubble. Adjust those values if you need more or less padding.

I want the box to resize based on the amount of text in the `Text` property

Alright, I understand now. We'll make the bubble size adapt dynamically based on the content of the text property, ensuring it fits the text well. Here's how you can achieve this:

1. **Adjust the size based on the text:** You can use the `Canvas.TextWidth` and `Canvas.TextHeight` methods to measure the dimensions of the text and adjust the control's dimensions accordingly.
2. **Set the minimum and maximum widths:** To ensure the bubble doesn't become too thin or too wide, you can define a minimum and maximum width for the bubble.

Here's the modified code:

```

delphi
// ... [rest of the code]

const
  MIN_BUBBLE_WIDTH = 100;
  MAX_BUBBLE_WIDTH = 250;
  PADDING = 10;

{ TBubbleText }

// ... [rest of the code]

procedure TBubbleText.ResizeBubble;

```

```

var
  TextWidth, TextHeight: Integer;
  NewWidth, NewHeight: Integer;
begin
  Canvas.Font.Assign(Font);
  TextWidth := Canvas.TextWidth(FLabel.Caption);
  TextHeight := Canvas.TextHeight(FLabel.Caption);

  NewWidth := TextWidth + 2 * PADDING;
  NewHeight := TextHeight + 2 * PADDING;

  if NewWidth < MIN_BUBBLE_WIDTH then
    NewWidth := MIN_BUBBLE_WIDTH
  else if NewWidth > MAX_BUBBLE_WIDTH then
    NewWidth := MAX_BUBBLE_WIDTH;

  Width := NewWidth;
  Height := NewHeight;
end;

procedure TBubbleText.SetText(const Value: string);
begin
  FLabel.Caption := Value;
  ResizeBubble;
  Invalidate;
end;

// ... [rest of the code]

```

The `ResizeBubble` method calculates the new width and height for the bubble based on the text's dimensions. The `SetText` method (setter for the `Text` property) has been modified to call `ResizeBubble` after updating the label's caption.

Note that the maximum and minimum widths (`MAX_BUBBLE_WIDTH` and `MIN_BUBBLE_WIDTH`) are arbitrary values. Adjust these constants as needed for your specific use case. The `PADDING` constant provides a buffer space around the text inside the bubble. You can adjust this too as per your requirements.

Does `Canvas.TextHeight` handle multiple lines of text?

No, `Canvas.TextHeight` does not handle multiple lines of text. It returns the height for a single line of text based on the currently selected font.

If you have a multi-line string and you want to calculate its height, you'll have to break the string into individual lines and sum up the heights for all lines. However, if you're using a `TLabel`, it's a bit more nuanced since `TLabel` handles word wrapping for you.

A more robust way to determine the height of multi-line text, especially in the context of the `TLabel` control, is to use the `DrawText` function with the `DT_CALCRECT` flag. This won't actually draw the text but will instead calculate the rectangle that the text would occupy.

Here's a function to calculate the height of multi-line text:

```
delphi
```

```

function CalculateTextHeight(const ACanvas: TCanvas; const Text: string;
MaxWidth: Integer): Integer;
var
  Rect: TRect;
begin
  Rect := Rect(0, 0, MaxWidth, 0);
  DrawText(ACanvas.Handle, PChar(Text), -1, Rect, DT_WORDBREAK or
DT_CALCRECT or DT_LEFT);
  Result := Rect.Bottom;
end;

```

You can use this function in the `ResizeBubble` procedure to correctly handle multi-line text:

```

delphi
procedure TBubbleText.ResizeBubble;
var
  TextWidth, TextHeight: Integer;
  NewWidth, NewHeight: Integer;
begin
  Canvas.Font.Assign(Font);

  TextWidth := Canvas.TextWidth(FLabel.Caption);
  TextHeight := CalculateTextHeight(Canvas, FLabel.Caption,
MAX_BUBBLE_WIDTH - 2 * PADDING);

  NewWidth := TextWidth + 2 * PADDING;
  NewHeight := TextHeight + 2 * PADDING;

  if NewWidth < MIN_BUBBLE_WIDTH then
    NewWidth := MIN_BUBBLE_WIDTH
  else if NewWidth > MAX_BUBBLE_WIDTH then
    begin
      NewWidth := MAX_BUBBLE_WIDTH;
      TextHeight := CalculateTextHeight(Canvas, FLabel.Caption, NewWidth - 2
* PADDING);
      NewHeight := TextHeight + 2 * PADDING;
    end;

  Width := NewWidth;
  Height := NewHeight;
end;

```

This way, if the text exceeds the maximum bubble width and wraps to a new line, the bubble height will be adjusted correctly.

Now can you add some padding around the `RoundRect` as specified by a `Padding` property of type `TPadding`

Certainly! The `TPadding` type, which is part of the VCL framework, allows you to specify padding around controls. To utilize it for the `TBubbleText` component, we'll:

1. Introduce a `Padding` property of type `TPadding`.
2. Adjust the drawing of the `RoundRect` and the positioning of the `Label` based on the values of the `Padding` property.

Here's the modified version of the code:


```

delphi
// ... [earlier code]

type
  TBubbleText = class(TCustomControl)
  private
    FLabel: TLabel;
    FBubbleType: TBubbleType;
    FPadding: TPadding;
    procedure SetText(const Value: string);
    function GetText: string;
    procedure SetBubbleType(const Value: TBubbleType);
    procedure SetPadding(const Value: TPadding);
  protected
    procedure Paint; override;
    procedure ResizeBubble;
    procedure Resize; override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property Text: string read GetText write SetText;
    property BubbleType: TBubbleType read FBubbleType write SetBubbleType
default btUser;
    property Padding: TPadding read FPadding write SetPadding;
    property Color;
    property Font;
    property Align;
    property Anchors;
    property Constraints;
    property Visible;
  end;

// ... [rest of the code]

{ TBubbleText }

constructor TBubbleText.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Width := 150;
  Height := 50;

  FLabel := TLabel.Create(Self);
  FLabel.Parent := Self;
  FLabel.AutoSize := False;
  FLabel.WordWrap := True;
  FLabel.Align := alClient;
  FLabel.Layout := tlCenter;
  FLabel.Transparent := True;

  FBubbleType := btUser; // Default type

  FPadding := TPadding.Create(Self);
end;

destructor TBubbleText.Destroy;
begin
  FPadding.Free;
  FLabel.Free;
  inherited;
end;

```

```

end;

procedure TBubbleText.Paint;
var
  Rect: TRectF;
begin
  inherited;

  Canvas.Pen.Color := Color;
  Canvas.Brush.Color := Color;

  case FBubbleType of
    btUser:
      begin
        Rect := RectF(FPadding.Left, FPadding.Top, Width * 0.75 -
FPadding.Right, Height - FPadding.Bottom);
        FLabel.SetBounds(FPadding.Left + 5, FPadding.Top, Trunc(Width *
0.75) - FPadding.Right - 10, Height - FPadding.Bottom);
        end;
    btOther:
      begin
        Rect := RectF(Width * 0.25 + FPadding.Left, FPadding.Top, Width -
FPadding.Right, Height - FPadding.Bottom);
        FLabel.SetBounds(Trunc(Width * 0.25) + FPadding.Left + 5,
FPadding.Top, Trunc(Width * 0.75) - FPadding.Right - 10, Height -
FPadding.Bottom);
        end;
      end;

  // Drawing the rounded rectangle as the bubble shape
  Canvas.RoundRect(Rect, 15, 15);
end;

procedure TBubbleText.SetPadding(const Value: TPadding);
begin
  FPadding.Assign(Value);
  ResizeBubble;
  Invalidate;
end;

// ... [rest of the code]

```

In this version, I've added the `Padding` property and made necessary adjustments in the `Paint` method to consider the padding values for drawing the bubble and positioning the label. Whenever the `Padding` property changes, it triggers a resize and repaint of the component to reflect the changes.

Remember to use the `Padding` property in the `ResizeBubble` method as well to accurately calculate the bubble size based on text and padding.

Can you make the color of the rect background settable as well as have a default background based on whether it is a `btUser` or `BtOther`

Absolutely. You can achieve this by introducing a new `BackgroundColor` property and setting default values based on the `BubbleType`.

1. **Add a BackgroundColor property:** This property will determine the color of the bubble.
2. **Set defaults for btUser and btOther:** For simplicity, let's set default colors as light blue for btUser and light gray for btOther.
3. **Adjust the Paint method:** Make sure the Paint method uses the BackgroundColor property for filling the bubble.

Here's how to incorporate these changes:

```
delphi
type
  TBubbleText = class(TCustomControl)
  private
    FLabel: TLabel;
    FBubbleType: TBubbleType;
    FBackgroundColor: TColor;
    FPadding: TPadding;
    procedure SetText(const Value: string);
    function GetText: string;
    procedure SetBubbleType(const Value: TBubbleType);
    procedure SetBackgroundColor(const Value: TColor);
    procedure SetPadding(const Value: TPadding);
  protected
    procedure Paint; override;
    procedure ResizeBubble;
    procedure Resize; override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property Text: string read GetText write SetText;
    property BubbleType: TBubbleType read FBubbleType write SetBubbleType
default btUser;
    property BackgroundColor: TColor read FBackgroundColor write
SetBackgroundColor;
    property Padding: TPadding read FPadding write SetPadding;
    property Font;
    property Align;
    property Anchors;
    property Constraints;
    property Visible;
  end;

{ TBubbleText }

constructor TBubbleText.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Width := 150;
  Height := 50;

  FLabel := TLabel.Create(Self);
  FLabel.Parent := Self;
  FLabel.AutoSize := False;
  FLabel.WordWrap := True;
  FLabel.Align := alClient;
  FLabel.Layout := tlCenter;
  FLabel.Transparent := True;
```

```

FBubbleType := btUser; // Default type

// Default background color based on BubbleType
if FBubbleType = btUser then
    FBackgroundColor := clSkyBlue
else
    FBackgroundColor := clSilver;

    FPadding := TPadding.Create(Self);
end;

// ... [rest of the code]

procedure TBubbleText.Paint;
var
    Rect: TRectF;
begin
    inherited;

    Canvas.Pen.Color := FBackgroundColor;
    Canvas.Brush.Color := FBackgroundColor;

    // ... [rest of the code for drawing based on BubbleType]

end;

procedure TBubbleText.SetBubbleType(const Value: TBubbleType);
begin
    if FBubbleType <> Value then
        begin
            FBubbleType := Value;
            // Set the default colors based on the BubbleType
            if FBubbleType = btUser then
                BackgroundColor := clSkyBlue
            else
                BackgroundColor := clSilver;
        end;
end;

procedure TBubbleText.SetBackgroundColor(const Value: TColor);
begin
    if FBackgroundColor <> Value then
        begin
            FBackgroundColor := Value;
            Invalidate; // Redraw when the color changes
        end;
end;

// ... [rest of the code]

```

With these changes, the bubble background color is set by the `BackgroundColor` property. When the `BubbleType` changes, the background color defaults to a specific color, but you can still customize it by setting the `BackgroundColor` property explicitly.

Can you make the text selectable?

Making text selectable inside a custom component requires a different approach. The built-in `TLabel` doesn't support text selection. Instead, we can use a `TMemo` or a `TRichEdit` to display

the text and allow selection. Both controls can be made borderless and read-only, allowing them to visually function similarly to `TLabel` but with selectable text.

I'll guide you on how to implement this with `TRichEdit`, which also gives you more formatting options if you want to expand functionality in the future.

Here's how you can modify the `TBubbleText` component to use a `TRichEdit`:

```
delphi
uses
    ..., RichEdit, ...

type
    TBubbleText = class(TCustomControl)
    private
        FEdit: TRichEdit; // Using TRichEdit instead of TLabel
        FBubbleType: TBubbleType;
        FBackgroundColor: TColor;
        FPadding: TPadding;
        procedure SetText(const Value: string);
        function GetText: string;
        // ... rest of the declarations
    end;

{ TBubbleText }

constructor TBubbleText.Create(AOwner: TComponent);
begin
    inherited Create(AOwner);
    Width := 150;
    Height := 50;

    FEdit := TRichEdit.Create(Self);
    FEdit.Parent := Self;
    FEdit.ReadOnly := True; // Ensure text is read-only
    FEdit.ScrollBars := ssNone; // Hide scrollbars
    FEdit.BorderStyle := bsNone; // Remove the border to make it look like a
    TLabel
    FEdit.WordWrap := True;

    // ... rest of the constructor

end;

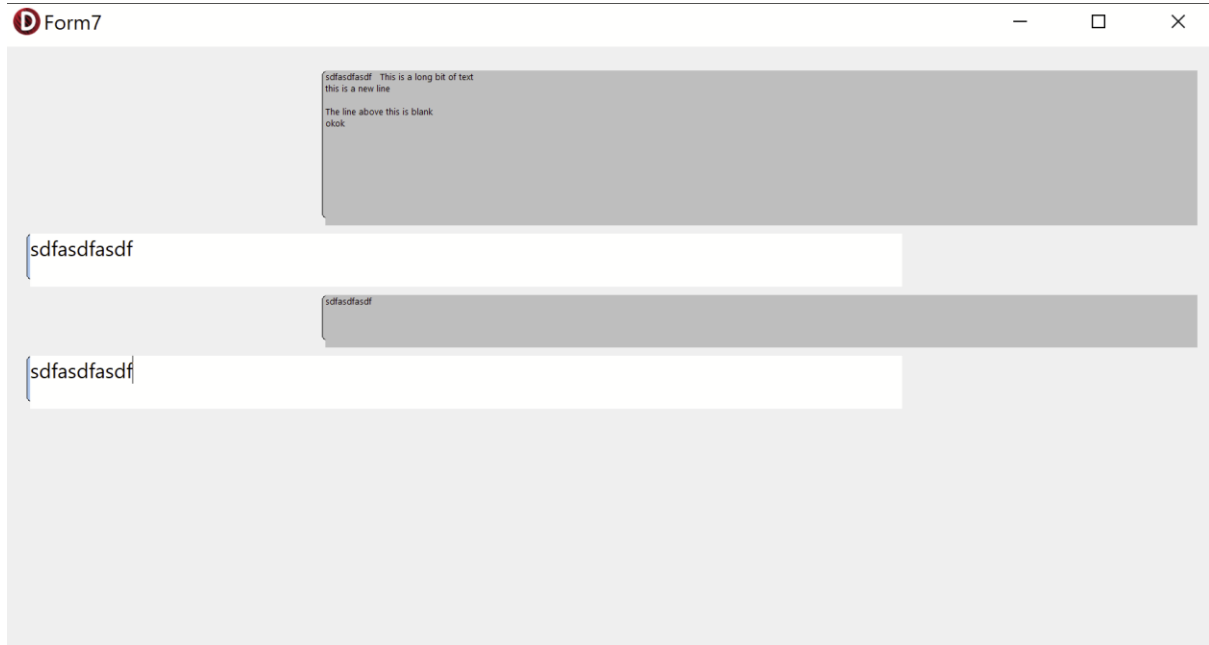
function TBubbleText.GetText: string;
begin
    Result := FEdit.Text;
end;

procedure TBubbleText.SetText(const Value: string);
begin
    FEdit.Text := Value;
    ResizeBubble;
end;

// ... rest of the code
```

You'll also need to adjust the positioning of the `TRichEdit` control in the `Paint` method and possibly the `ResizeBubble` method, similar to how you handled the `TLabel`.

With this approach, the text inside the bubble will be selectable, and you can even allow copying to the clipboard if desired.



Here is the sample output using the richedit. The background color doesn't come through properly and it loses the rounded edges of the rectangle

I see. The `TRichEdit` has its own way of drawing content which may override our custom drawing. This means the background color and the rounded edges you set for the custom control might not be reflected in the `TRichEdit` itself.

To overcome this issue, we need to set the `TRichEdit` to be transparent and let the custom control handle the painting. This way, the custom control will handle the drawing of the background and the rounded edges, while the `TRichEdit` will only handle the text.

Here's what you can do:

1. **Set `TRichEdit` to be transparent:** You can achieve this by setting its `Transparent` property to `True`.
2. **Resize and reposition the `TRichEdit`:** You should also resize and reposition the `TRichEdit` to be within the bounds of the custom control's rounded rectangle, considering the padding.
3. **Custom draw the background and rounded edges:** In the custom control's `Paint` method, you should handle the drawing of the background and the rounded edges.

Here's a code outline on how to modify the `TBubbleText` class:

```
delphi
constructor TBubbleText.Create(AOwner: TComponent);
begin
```

```

// ... your existing initialization code ...

FEdit := TRichEdit.Create(Self);
FEdit.Parent := Self;
FEdit.Transparent := True; // Make it transparent
FEdit.ReadOnly := True;
FEdit.ScrollBars := ssNone;
FEdit.BorderStyle := bsNone;
FEdit.WordWrap := True;

// ...
end;

procedure TBubbleText.ResizeBubble;
var
    TxtWidth, TxtHeight: Integer;
begin
    // ... existing code ...

    // Adjust the position and size of TRichEdit to fit within the rounded
    rectangle
    FEdit.Left := FPadding.Left;
    FEdit.Top := FPadding.Top;
    FEdit.Width := TxtWidth - FPadding.Left - FPadding.Right;
    FEdit.Height := TxtHeight - FPadding.Top - FPadding.Bottom;
end;

procedure TBubbleText.Paint;
var
    Rect: TRectF;
begin
    Canvas.Pen.Color := FBackgroundColor;
    Canvas.Brush.Color := FBackgroundColor;

    // Draw the rounded rectangle considering the padding
    Rect := RectF(FPadding.Left, FPadding.Top, Self.Width - FPadding.Right,
Self.Height - FPadding.Bottom);

    // Adjust position for left or right alignment
    if FBubbleType = btOther then
        OffsetRect(Rect, Self.Width - TxtWidth - (2 * FPadding.Right), 0)
    else
        OffsetRect(Rect, FPadding.Left, 0);

    Canvas.RoundRect(Rect, 10, 10);

    // Since TRichEdit is transparent, it will show the rounded rectangle
    beneath it.
end;

```

With these changes, the `TRichEdit` will show the text within the boundaries of the rounded rectangle drawn by the custom control, giving the desired appearance.