How to paint on a TControlCanvas in a TMemo

Answer 1

Create a new component derived from *TMemo* and override its drawing. Something like this:

```
type
  TMyMemo = class(TMemo)
  protected
    procedure WMPaint(var Message: TWMPaint); message WM_PAINT;
  end;

procedure TMyMemo.WMPaint(var Message: TWMPaint);
var
  MCanvas: TControlCanvas;
  DrawBounds: TRect;
begin
  inherited;
  MCanvas := TControlCanvas.Create;
  DrawBounds := ClientRect;
  try
    MCanvas.Control := Self;
    with MCanvas do
    begin
      Brush.Color := clBtnFace;
      FrameRect( DrawBounds );
      InflateRect( DrawBounds, - 1, - 1);
      FrameRect( DrawBounds );
      FillRect ( DrawBounds );
      MoveTo ( 33, 0 );
      Brush.Color := clWhite;
      LineTo ( 33, ClientHeight );
      PaintImages;
    end;
  finally
    MCanvas.Free;
  end;
end;
```

The *PaintImages* procedure draws images on the *TMemo*'s canvas.

```
procedure TMyMemo.PaintImages;
var
  MCanvas: TControlCanvas;
  DrawBounds: TRect;
  i, j: Integer;
  OriginalRegion: HRGN;
  ControlDC: HDC;
begin
  MCanvas := TControlCanvas.Create;
  DrawBounds := ClientRect;
  try
    MCanvas.Control := Self;
    ControlDC := GetDC ( Handle );
    MCanvas.Draw(0, 1, Application.Icon);
  finally
    MCanvas.Free;
  end;
end;
```

Tip author unknown

Answer 2

Basically you will need to intercept *WM_ERASEBKGND* and *WM_PAINT* messages. Let's say you have a *TImage* control the same size as your *TMemo* holding a bitmap that you want to use as your background.

Let's assume you have this hooked in a *TImage* field called *FImage* available in your memo component code. The following should give you a good start:

In your class definition for *TMyMemo*:

```
procedure WMEraseBkGnd(var Message: TWMEraseBkGnd); message WM_ERASEBKGND;
procedure WMPaint(var Message: TWMPaint); message WM_PAINT;

{...}

procedure TMyMemo.WMEraseBkGnd(var Message: TWMEraseBkGnd);
begin
  {assuming we get a good DC in Message - you should check this of course}
  BitBlt(Message.dc, 0, 0, Width, Height, FImage.Canvas.Handle, 0, 0, SRCCOPY);
  Message.Result := - 1;
end;

procedure TMyMemo.WMPaint(var Message: TWMPaint);
var
  bm: TBitmap;
  dc: HDC;
  hDummy: HWND;
  i: integer;
  tm: TEXTMETRIC;
  Y: integer;
begin
  bm := TBitmap.Create;
  try
    bm.Width := Width;
    bm.Height := Height;
    Perform(WM_ERASEBKGND, bm.Canvas.Handle, 0); {always in this simple example}
    bm.Canvas.Font.Assign(Font);
    GetTextMetrics(bm.Canvas.Handle, tm);
    SetBkMode(bm.Canvas.Handle, TRANSPARENT);
    Y := 0;
    for i := 0 to Lines.Count - 1 do
    begin
      bm.Canvas.TextOut(0, Y, Lines[i]);
      Inc(Y, tm.tmHeight);
    end;
    dc := GetDeviceContext(hDummy);
    BitBlt(dc, 0, 0, Width, Height, bm.Canvas.Handle, 0, 0, SRCCOPY);
    ReleaseDC(hDummy, dc);
  finally
    bm.Free;
  end;
  Message.Result := 0;
end;
```

Note that this is only good for displaying transparently. Editing is another story. What I do is call the inherited behavior when I'm editing (so no transparency while typing). Obviously this example has no error checking. Also, the *Message* parameter for *WM_PAINT* may contain a device context to use in lieu of *GetDeviceContext*. The text always draws at `X = 0`

Tip by Jon T Camp

Original resource:	The Delphi Pool
Author:	Anon & Jon T Camp
Added:	2013-01-27
Last updated:	2013-01-27