

Delphi 高级组件开发指南

第一篇

作者：金海龙（软件工程师）

简 介

跟流行软件中的控件相比，TButton没有圆角边框、没有渐变背景图、没有……

总之，TButton没有吸引人的地方，通过对TButton的改进，制造一个能吸引人的按钮，

从而提升软件的界面友好（提高竞争力），这就是：TJHLButton，我写了这篇技术文章，

本文包含如下主要内容：

一、学会在TButton表面绘图

继承自TButton的TJHLButton，在WMPaint()中改变了按钮的老风格；

二、按钮的渐变背景图

我编写的渐变图绘制函数，通过本文共享出来；

三、按钮的文字

多行文字的显示，只要通过我的函数就能轻松实现；

四、按钮的圆角边框

如何把按钮变成圆角，看看这儿的技术资料；

五、按钮的圆角外形

把按钮变成圆角，不是轻松的，不看本文，普通程序员至少需要1个星期才能攻破。

完全掌握本文技术大概需要7天的时间，如果读者自己独自攻破本文所涉及的技术，那么至少需要1个月的时间，但如果读了本文，就只需要7天。

声明：本文只发布在豆丁网，禁止转载。

如果本文的下载量没有达到100次，那么我就不会继续写更精彩的，

所以大家不要使用‘Flash 提取’一类的软件非法下载。

本文所有代码已经在Win2000+Delphi6平台上调试通过，

组件可以运行在Delphi6或更高版本。



引 言

借助于Delphi第三方组件，可以使自己的软件界面更美观，第三方组件扩展了Delphi，完成了Delphi无法达到的功能，所以很多人热衷于开发第三方插件。

但没有多少人能开发出高质量的插件，

我是一名软件工程师，要想知道我的国际知名程度，

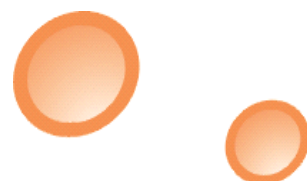
Google搜索：金海龙

经验不是一天能积累起来的，但若没有经验就开发不出高水准的第三方组件。

希望有人能记住我（只要记住我的名字，就可以通过Google搜到更多的技术文档），

于是我决定给这个按钮取的名字中间加上“JHL”三个字母，这是我中文名字的汉语拼音字头缩写，即：JHL=Jin Hailong 。

从Delphi6那个年代起，我就开始开发第三方组件，越来越多的电脑毕业生开始“靠开发软件赚钱”，第三方插件弥补了Delphi功能上的不足，使一个刚入门的程序员在短时间内提高了编程能力。没有第三方组件，一个刚毕业的学生绝不可能设计出高超的软件。所以，许多人开始关注‘开发Delphi插件赚钱’。作为一名高手，把自己的编程能力做成插件，共享给成千上万万的程序员，可以赚名，可以谋利。



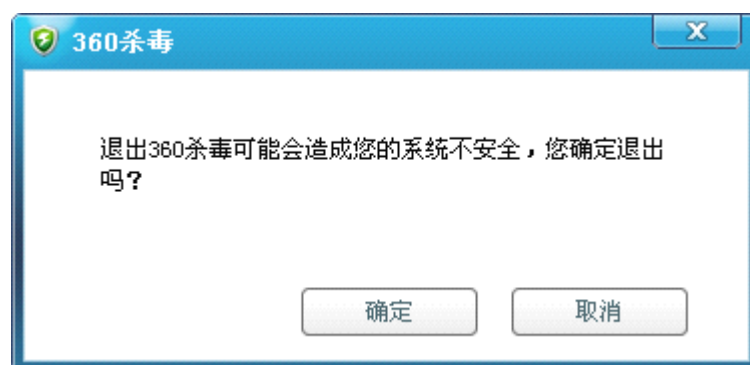
许多个人开发者都想涉足‘第三方插件市场’，但不知如何开始。

在我看来，只要掌握一个插件的开发流程，就能开发出更多的插件，这就叫：举一反三、一通百通

本文以一个按钮的开发过程为例，讲解控件界面设计的秘密技术。不论是按钮，还是文本框，都具有背景、文字、边框三个要素，因此只要掌握了按钮的界面设计，就等同于掌握了文本框的界面设计。

一、学会在TButton表面绘图

在开始设计自己的按钮之前，先看看下面软件中的按钮，想想它们的共同点都有哪些：



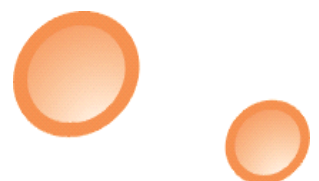


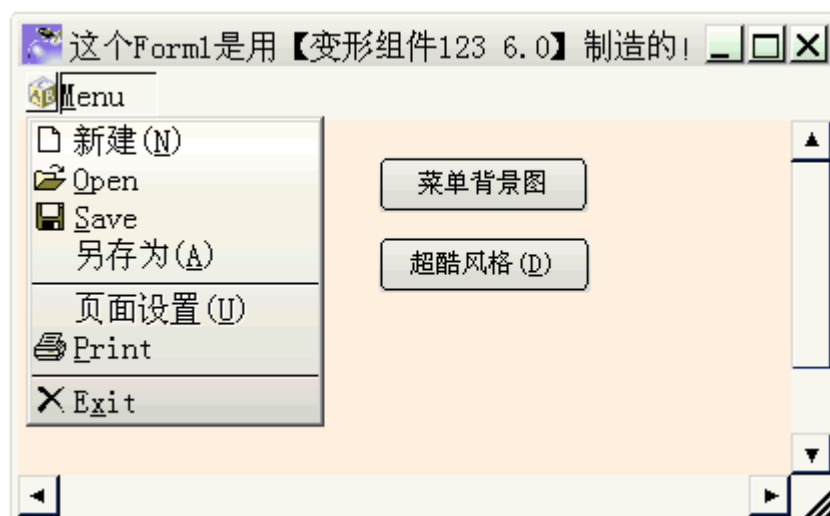
优秀软件中的按钮都具备这些共同点：渐变背景、圆角边框

每一个按钮都是由背景图、文字和圆角边框组成。

我曾经为Delphi6写了第三方插件：变形组件123，

可以免费下载源代码，其中的部分按钮如下图所示：

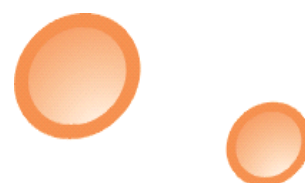




由于Delphi6没有提供在画布上画渐变图的函数，对于很多程序员来说还得自己编写代码（很不方便），

我开发的《变形组件123 6.0》弥补了Delphi6的不足，本文的画渐变图的函数，就是在其中节选并稍加修改。

画一个按钮的基本步骤是：



- 1、画渐变背景；
- 2、写文字；
- 3、画圆角边框；
- 4、把矩形按钮变成圆角；

下面启动你的Delphi，新建一个组件，我们就要在这个新的组件中编写代码，
信息如下：

单元名： JHLButton

继承自：TButton

类名： TJHLButton

组件夹： DelphiDemos

该文件内容如下：

```
unit JHLButton;

interface

uses
  Windows, Messages, SysUtils, Classes, Controls, StdCtrls;

type
  TJHLButton = class(TButton)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
```



```

        { Public declarations }
published
        { Published declarations }
end;

procedure Register;

implementation

procedure Register;
begin
    RegisterComponents(' DelphiDemos', [TJHLButton]);
end;

end.

```

如果把组件安装以后，就可以在“DelphiDemos”组件夹下找到按钮组件TJHLButton。

每个按钮都有“Invalidate;”过程，当调用按钮的这个过程后，系统自动调用“WMPaint();”，我们就要在“WMPaint();”中编写代码，从而达到在按钮表面绘图的目的。

由于要在按钮表面绘图，所以需要用到Graphics.pas，这就决定着要把“Graphics”添加到uses部分，修改后的uses部分如下：

```

uses
    Windows, Messages, SysUtils, Classes, Controls, StdCtrls, Graphics;

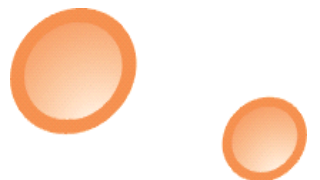
```

接着要在Private部分添加函数：WMPaint(); 并且在implementation部分编写这个函数的内容，如下所示：

```

.....
private
    { Private declarations }
    procedure WMPaint(var Message: TWMPaint); message WM_PAINT;
.....

```



implementation

```
procedure TJHLButton.WMPaint(var Message: TWMPaint);
var b:TBitmap; c:TControlCanvas;
begin
  inherited;
  //0. 准备一张要画到按钮表面的位图
  b:=TBitmap.Create;
  b.Width:=width; b.Height:=height;//这张图片的尺寸要和按钮的尺寸一样
  //1. 在b上画按钮表面的第一部分：背景图

  //2. 在b上画按钮表面的第二部分：文字

  //3. 在b上画按钮表面的第三部分：边框

  //4. 在b上画按钮表面的第四部分：表示按钮拥有焦点的矩形虚线框

  //5. 把b画到按钮的表面
  c:=TControlCanvas.Create;
  c.Control:=self;
  c.Draw(0, 0, b);
  c.Free;
  //6. 释放b所占用的内存空间
  b.Free;
end;

.....
```

如果对函数“WMPaint()”比较陌生，只要通过[google搜索](#)就能得到很多相关资料，由于“WMPaint()”属于入门级的内容，所以本文不再重述。

上面“WMPaint()”中代码的大概意思是说：在内存中制作按钮表面图片，然后画到按钮表面。

“0-4”步是制作按钮表面图片，第5步把制作好的图片画到按钮表面。



注意观察“1-4”步，其中有空白的行，下面我们就用代码填充这些空白。

如果按照上面的内容编写TJHLButton组件，当你用鼠标拖动一个到Form1中，你会发现这个按钮的表面是白色的，
那是因为还没有画边框、文字，但这意味着我们已经成功屏蔽了TButton默认的灰底黑字的windows风格，可以编写我们自己的按钮了。

在运行时，当你用鼠标点击按钮，或者按下空格键，就会发现默认的灰底黑字又出现了，这就意味着需要合理地调用“Invalidate;”过程，从而正确调用WMPaint()

需要按照下面的代码编写JHLButton.pas文件：

```
.....
private
    { Private declarations }
.....

procedure CMEnabledChanged(var Message: TMessage); message CM_ENABLEDCHANGED;
procedure CMFontChanged(var Message: TMessage); message CM_FONTCHANGED;
Protected
    { Protected declarations }
    procedure KeyDown(var Key: Word; Shift: TShiftState); override;
    procedure KeyUp(var Key: Word; Shift: TShiftState); override;
    procedure MouseDown(Button: TMouseButton; Shift: TShiftState;
        X, Y: Integer); override;
    procedure MouseMove(Shift: TShiftState; X, Y: Integer); override;
    procedure MouseUp(Button: TMouseButton; Shift: TShiftState;
        X, Y: Integer); override;

public
.....
```



implementation

```
procedure TJHLButton.CMEnabledChanged(var Message: TMessage);
```

```
begin
```

```
    inherited;
```

```
    Invalidate;//更改enabled属性时，重新显示表面图片
```

```
end;
```

```
procedure TJHLButton.CMFontChanged(var Message: TMessage);
```

```
begin
```

```
    inherited;
```

```
    Invalidate;//更改Font时，重新显示表面图片
```

```
end;
```

```
procedure TJHLButton.KeyDown(var Key: Word; Shift: TShiftState);
```

```
begin
```

```
inherited; if key=32 then Invalidate;//按下空格键时，重新显示表面图片
```

```
end;
```

```
procedure TJHLButton.KeyUp(var Key: Word; Shift: TShiftState);
```

```
begin
```

```
inherited; if key=32 then Invalidate;//在释放空格键时，重新显示表面图片
```

```
end;
```

```
procedure TJHLButton.MouseDown(Button: TMouseButton; Shift: TShiftState;
```

```
    X, Y: Integer);
```

```
begin
```

```
inherited; Invalidate;//在按下鼠标左键时，重新显示表面图片
```

```
end;
```

```
procedure TJHLButton.MouseMove(Shift: TShiftState; X, Y: Integer);
```

```
begin
```

```
inherited; if (ssLeft in Shift) or (ssRight in Shift) then Invalidate;
```

```
//按住鼠标左键在按钮表面移动时，重新显示表面图片
```

```
end;
```

```
procedure TJHLButton.MouseUp(Button: TMouseButton; Shift: TShiftState;
```

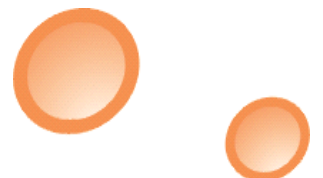
```
    X, Y: Integer);
```

```
begin
```

```
inherited; Invalidate;//在释放鼠标左键时，重新显示表面图片
```

```
end;
```

```
.....
```



可以看到上面的代码中，新添加了7个过程，它们都调用了“`Invalidate;`”过程，也就相当于间接调用`WMPaint()`。

二、按钮的渐变背景图

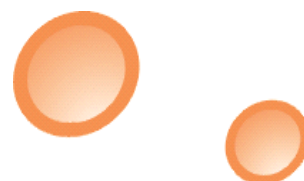
Delphi6没提供画渐变图的函数，自己去编写，对于新手来说，至少耗费十多个小时。

我编写了一个专门用于制作按钮渐变图的函数：

```
Function BtnJB(Color:TColor;BtnHeight:integer=25;Step:integer=3):TBitmap;  
“Color” 用于指定渐变的起始颜色;  
“BtnHeight” 是返回位图的高度;  
“Step” 用于指定渐变的大小，如果值小于0，那么从下向上渐变；如果值大于0，那么从上  
向下渐变;
```

先在JHLButton.pas的implementation部分添加如下代码：

```
Function GetRedFrom(C:TColor):byte;  
begin  
//读取C中的红色值  
Result:=getRvalue(ColorToRGB(C));  
end;  
  
Function GetGreenFrom(C:TColor):byte;  
begin  
//读取C中的绿色值  
Result:=getGvalue(ColorToRGB(C));  
end;  
  
Function GetBlueFrom(C:TColor):byte;  
begin  
//读取C中的蓝色值  
Result:=getBvalue(ColorToRGB(C));  
end;
```



```

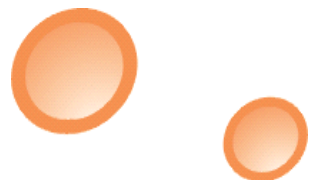
procedure LimitRGB(var I:Integer);
begin
//i是red, green, blue三色中的一色，此函数使i不超过0-255的范围
if i>255 then i:=255;
if i<0 then i:=0;
end;

Function AddRGBVal(C:TColor;I:Integer;ChangeRGB:String='redgreenblue'):TColor;
VAR r,g,b:integer; S:String;
begin
//累加C中的red, green, blue，此函数使每个色值不超过0-255的范围
s:=lowercase(changergb);
r:=getredfrom(C); G:=getGREENfrom(C); B:=getBLUEfrom(C);
if pos('red',S)<>0 THEN INC(R,I); LimitRGB(R);
if pos('green',S)<>0 THEN INC(G,I); LimitRGB(G);
if pos('blue',S)<>0 THEN INC(B,I); LimitRGB(B);
result:=rgb(r,g,b);
end;

Function CaptionBarJB(BeginColor:TColor=clred;Num:integer=10;Step:integer=5;
ChangeRGB:String='redgreenblue'):TBitmap;
var B:TBitmap; C1:TColor;
i,co:Integer;
begin
//标题栏渐变背景图
B:=TBitmap.Create; B.PixelFormat := pf24Bit;
B.Width:=1; B.Height:=num;
CO:=Num-1;
for i:=0 to co do
begin
c1:=AddRGBVal(BeginColor,i*abs(step),ChangeRGB);
if step<0 then
b.Canvas.Pixels[0,co-i]:=c1
else
b.Canvas.Pixels[0,i]:=c1;
end;
Result:=B;
end;

Function BtnJB(Color:TColor;BtnHeight:integer=25;Step:integer=3):TBitmap;
begin
result:=captionbarjb(Color,BtnHeight,Step);
end;

```



然后再按照下面的意图修改WMPaint()中的代码:

```
procedure TJHLButton.WMPaint(var Message: TWMPaint);
var b:TBitmap; c:TControlCanvas;
begin inherited;
//0. 准备一张要画到按钮表面的位图
b:=TBitmap.Create;
b.Width:=width; b.Height:=height;//这张图片的尺寸要和按钮的尺寸一样
//1. 在b上画按钮表面的第一部分: 背景图
b.Canvas.StretchDraw(ClientRect, BtnJB($00C08080, 25, 3));//$00F9E3D5
//2. 在b上画按钮表面的第二部分: 文字
.....
```

仔细看上面的代码, 其中的“b.Canvas.StretchDraw()”一行, 就是运用BtnJB()函数的方法。

为何要用Canvas的拉伸函数画渐变图呢?

仔细观察渐变图, 可以发现‘同一行中的颜色值都相等, 同一列中的颜色值都是递增的’, 所以可以想到: 用TImage装入一幅宽1高25的位图, 只要把属性Stretch设置成True, 那么就能看到矩形图, 而不是看到一条线。

‘BtnJB(clbtnface, 25, 3)’的意思是: 制作一张渐变位图, 要求位图高25, 宽度是1, 从clbtnface开始渐变, 颜色值每次累加3;

‘BtnJB(clbtnface, 25, -3)’的意思是: 制作一张渐变位图, 要求位图高25, 宽度是1, 从clbtnface开始渐变, 颜色值每次累加3, 并且把渐变图上下颠倒;

有些流行软件中的按钮, 用户把鼠标移到上面就显示不同的背景, 通常的做法是: 把原来的渐变图上下颠倒;

即: 在OnMouseLeave发生时在按钮表面显示一张从上向下的渐变图,



在OnMouseEnter发生时在按钮表面显示一张从下向上的渐变图(颠倒原来的背景图)。

要想知道选择哪种颜色作为背景色较好,可以参考国际优秀软件。

把其中颜色最深的部分作为BtnJB()的颜色参数。

我曾经开发的Delphi6第三方组件《变形组件123 6.0》中有如下的渐变图组件:



大家有兴趣,欢迎从网上下载(用Google搜索就能找到)。

其中的绘图函数,很方便初学者借鉴。

三、按钮的文字



由于Delphi6提供的写字函数，只能写单行文字，并且对于其中的特殊字符‘&’号，不能很好显示，给编程带来很大不便；再加上Win32 API提供的写字函数又很不好用，

我编写了一个专门用于在button表面写字的函数：

```
procedure DrawButtonCaption(  
Canvas:TCanvas;  
R1:TRect;  
S:TStrings;  
TopAlign:boolean=false;  
LeftAlign:boolean=true;  
Enabled:boolean=true;  
DisabledColor:TColor=clwhite);
```

虽然参数很多，但只要有耐心看下去，你会很喜欢这个函数。

大概功能：左对齐写字符串，多余的文字将被隐藏；

各个参数的意思如下：

前三个参数是说‘把S中的文字写到Canvas的R1区域’；

参数TopAlign指定：在垂直方向是否靠上对齐写文字；

参数LeftAlign指定：是否左对齐写文字；

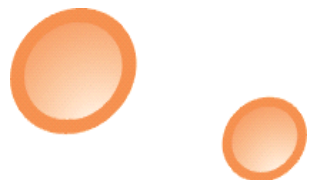
如果Enabled等于false，那么参数disabledcolor就会起作用，

显示的文字将会带有阴影，阴影的颜色由disabledcolor决定。

如果按钮的enabled属性为假，那么就需要同时设置最后两个参数。

先在implementation部分添加如下代码：

```
function CalTStringsHeight(Canvas:TCanvas;S:TStrings):Integer;  
begin  
//把s写到画布canvas上，若要垂直居中对齐，需要计算文字的高度，这个函数用来计算S  
里文字的高度  
Result:=S.Count*Canvas.TextHeight('金');  
end;  
  
function VAlignRect(R1:TRect;RctHeight:integer):TRect;  
var i:integer; r:TRect;  
begin  
//计算居中对齐的区域  
R:=r1;  
i:=abs(R1.Bottom-R1.Top);  
end;
```




```

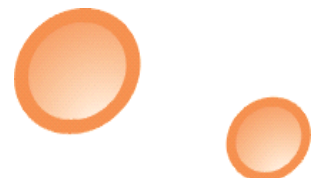
if RctHeight<i then
R:=Rect(r1.Left,r1.Top+round((i-RctHeight)/2),r1.Right,r1.Bottom);
result:=r;
end;

function CalTStringsWidth(Canvas:TCanvas;S:TStrings):Integer;
var i,co,ii,r:integer;
begin
//把s写到画布canvas上，若要垂直居中对齐，需要计算文字的高度，这个函数用来计算S
里文字的高度
co:=s.Count-1; r:=0;
for i:=0 to co do
begin
ii:=Canvas.TextWidth(s[i]);
if ii>r then r:=ii;
end; //over for
result:=r;
end;

function HAlignRct(R1:TRect;RctWidth:integer):TRect;
var i:integer; r:TRect;
begin
//计算居中对齐(水平)的区域
R:=r1;
i:=abs(R1.Right-R1.Left);
if RctWidth<i then
R:=Rect(r1.Left+round((i-RctWidth)/2),r1.Top,r1.Right,r1.Bottom);
result:=r;
end;

procedure
DrawButtonCaption(Canvas:TCanvas;R1:TRect;S:TStrings;TopAlign:boolean=false;
LeftAlign:boolean=true;Enabled:boolean=true;DisabledColor:TColor=clwhite);
var uFormat:cardinal; R2,R3:TRect; C:TColor;
begin
{左对齐写字符串, 多余的将被隐藏;
参数TopAlign指定: 在垂直方向是否靠上对齐
如果Enabled等于false, 那么参数disabledcolor就会起作用,
显示的文字将会带有阴影, 阴影的颜色由disabledcolor决定。}
R2:=R1;
if topalign then
uFormat:=DT_LEFT or DT_Bottom
else
begin

```



```

uFormat:=DT_LEFT;
R2:=VAlignRect(R1, CalTStringsHeight(Canvas, S));
end;
if leftalign=false then R2:=HAlignRect(R2, CalTStringsWidth(Canvas, S));
if not Enabled then
Begin
C:=Canvas.Font.Color;
Canvas.Font.Color:=DisabledColor;
R3:=Rect(R2.Left+1, R2.Top+1, R2.Right, R2.Bottom);
DrawText(Canvas.Handle, PChar(S.Text), -1, R3, uFormat);
Canvas.Font.Color:=C;
End;
DrawText(Canvas.Handle, PChar(S.Text), -1, R2, uFormat);//这是Win32 API提供的函数
end;

```

然后再按照下面的样子改写WMPaint()代码:

```

procedure TJHLButton.WMPaint(var Message: TWMPaint);
var ..... cap:TStrings;
begin
.....
//2. 在b上画按钮表面的第二部分: 文字
cap:=TStringlist.Create;
cap.Text:=caption;
b.Canvas.Brush.Style:=bsclear;
DrawButtonCaption(b.Canvas, ClientRect, cap,
false, false, self.Enabled, clwhite);
cap.Free;
//3. 在b上画按钮表面的第三部分: 边框
.....

```

上面的代码用来在按钮上写字。

四、按钮的圆角边框



流行软件中的按钮大都是圆角，圆角分成2种，

一是正圆角，另外一种是椭圆角，

由于Delphi没有提供画圆角矩形的函数，所以对编程造成困难。

为了攻克这个难关，初学者至少要1个月的时间，

看了下面的文章，就省力气了。

我编写了一个函数，专门用来画圆角边框：

```
Procedure draw2Rectangle(Canvas:TCanvas;A,C:TPoint;Radius:Integer=3;  
Color1:TColor=clwhite;Color2:TColor=clblack;Color3:TColor=clwhite;Color4:TColor  
=clblack;  
Visible:String=' 12345678');
```

由参数“A,C”可以定义Canvas上的一块矩形区域，

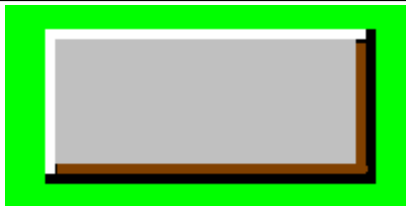
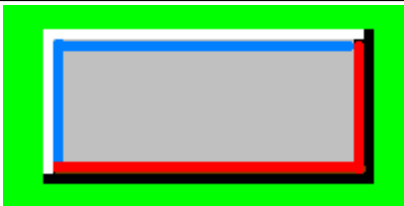
这个函数就在这个矩形区域中画圆角边框。

这个矩形区域左上角的坐标是A,右下角的坐标是C，圆角的大小

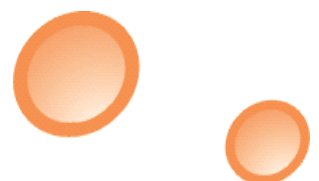
由参数radius确定，radius的取值范围是：0-19之间的奇数；

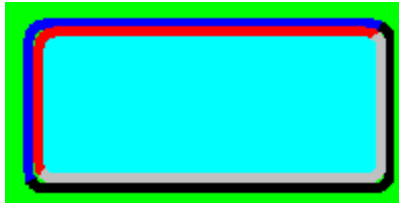
如果Radius=0，那么绘画矩形边框；如果大于0，那么绘画圆角矩形。

函数名为何叫‘draw2Rectangle’呢？如果你用放大镜看TButton的边框，就会发现它是由两个矩形组成的，如下图所示：

	
外层矩形由白色和亮黑色组成； 内层矩形由灰色和深黑色组成；	外层矩形由白色和亮黑色组成； 内层矩形由蓝和红色组成；

上面所示的是矩形边框，圆角边框也是由内外两个圆角矩形组成，





如上所示，圆角矩形是由内外两层组成，外层由蓝黑两色组成，内层由红灰两色组成。如果缩小了，就能体现出立体感。

在函数draw2Rectangle()的参数中，Color1和Color2控制外层矩形的颜色；Color3和Color4控制内层矩形的颜色；

下面我们开始在按钮表面画圆角边框，

先在implementation部分添加如下代码：

```
procedure DrawLine(WhichCanvas:TCanvas;p1,p2:TPoint);
BEGIN
//从p1到p2画一条线段
with WhichCanvas do
begin
MoveTo(p1.x,p1.y);
LineTo(p2.x,p2.y);
end;
END;

Procedure draw2ColorRect(Canvas:TCanvas;A,C:TPoint;Color1:TColor=clwhite;
Color2:TColor=clblack;Visible:String='1234');
VAR P1,P2:TPoint; l,t,r,b:Integer; C1:TColor;
Begin
//画两色矩形边框
C1:=Canvas.Pen.Color;
//
l:=A.X; T:=A.Y; R:=C.X; B:=C.Y;
//
Canvas.Pen.Color:=Color1;
if Color1<>clnone then
BEGIN
if pos('1',Visible)<>0 then
begin
P1:=Point(l,t); P2:=Point(l,b);
drawline(Canvas,P1,P2);
```

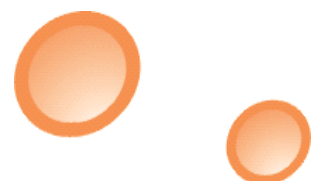
```

end;
if pos('4', Visible) <> 0 then
begin
P1:=Point(1, t); P2:=Point(r, t);
drawline(Canvas, P1, P2);
end;
END;
//-----
Canvas.Pen.Color:=Color2;
if Color2 <> clNone then
BEGIN
if pos('2', Visible) <> 0 then
begin
P1:=Point(1, b); P2:=Point(r, b);
drawline(Canvas, P1, P2);
end;

if pos('3', Visible) <> 0 then
begin
P1:=Point(r, b); P2:=Point(r, t-1);
drawline(Canvas, P1, P2);
end;
END;
//
Canvas.Pen.Color:=C1;
End;

function RC3Radius:TPoints;
var P:TPoints;
begin
setlength(p, 4);
P[0]:=Point(0, 3);
P[1]:=Point(1, 3);
P[2]:=Point(2, 2);
P[3]:=Point(3, 0);
result:=P;
end;
//end:30
//begin:30
function RC5Radius:TPoints;
var P:TPoints;
begin
setlength(p, 6);
P[0]:=Point(0, 5);

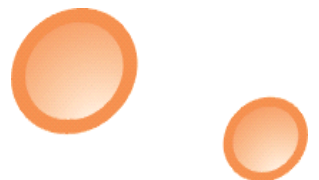
```



```

P[1]:=Point(1, 5);
P[2]:=Point(2, 5);
P[3]:=Point(3, 4);
P[4]:=Point(4, 3);
P[5]:=Point(5, 0);
result:=P;
end;
//end:30
//begin:30
function RC7Radius:TPoints;
var P:TPoints;
begin
setlength(p, 8);
P[0]:=Point(0, 7);
P[1]:=Point(1, 7);
P[2]:=Point(2, 7);
P[3]:=Point(3, 6);
P[4]:=Point(4, 6);
P[5]:=Point(5, 5);
P[6]:=Point(6, 3);
P[7]:=Point(7, 0);
result:=P;
end;
//end:30
//begin:30
function RC9Radius:TPoints;
var P:TPoints;
begin
setlength(p, 10);
P[0]:=Point(0, 9);
P[1]:=Point(1, 9);
P[2]:=Point(2, 9);
P[3]:=Point(3, 8);
P[4]:=Point(4, 8);
P[5]:=Point(5, 7);
P[6]:=Point(6, 7);
P[7]:=Point(7, 5);
P[8]:=Point(8, 3);
P[9]:=Point(9, 0);
result:=P;
end;
//end:30
//begin:30
function RC11Radius:TPoints;

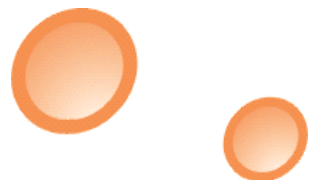
```



```

var P:TPoints;
begin
  setlength(p, 12);
  P[0]:=Point(0, 11);
  P[1]:=Point(1, 11);
  P[2]:=Point(2, 11);
  P[3]:=Point(3, 11);
  P[4]:=Point(4, 10);
  P[5]:=Point(5, 10);
  P[6]:=Point(6, 9);
  P[7]:=Point(7, 8);
  P[8]:=Point(8, 7);
  P[9]:=Point(9, 6);
  P[10]:=Point(10, 4);
  P[11]:=Point(11, 0);
  result:=P;
end;
//end:30
//begin:30
function RC13Radius:TPoints;
var P:TPoints;
begin
  setlength(p, 14);
  P[0]:=Point(0, 13);
  P[1]:=Point(1, 13);
  P[2]:=Point(2, 13);
  P[3]:=Point(3, 13);
  P[4]:=Point(4, 12);
  P[5]:=Point(5, 12);
  P[6]:=Point(6, 12);
  P[7]:=Point(7, 11);
  P[8]:=Point(8, 10);
  P[9]:=Point(9, 9);
  P[10]:=Point(10, 8);
  P[11]:=Point(11, 7);
  P[12]:=Point(12, 4);
  P[13]:=Point(13, 0);
  result:=P;
end;
//end:30
//begin:30
function RC15Radius:TPoints;
var P:TPoints;
begin

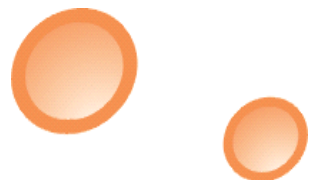
```



```

setlength(p, 16);
P[0]:=Point(0, 15);
P[1]:=Point(1, 15);
P[2]:=Point(2, 15);
P[3]:=Point(3, 15);
P[4]:=Point(4, 14);
P[5]:=Point(5, 14);
P[6]:=Point(6, 14);
P[7]:=Point(7, 13);
P[8]:=Point(8, 13);
P[9]:=Point(9, 12);
P[10]:=Point(10, 11);
P[11]:=Point(11, 10);
P[12]:=Point(12, 9);
P[13]:=Point(13, 7);
P[14]:=Point(14, 4);
P[15]:=Point(15, 0);
result:=P;
end;
//end:30
//begin:30
function RC17Radius:TPoints;
var P:TPoints;
begin
setlength(p, 18);
P[0]:=Point(0, 17);
P[1]:=Point(1, 17);
P[2]:=Point(2, 17);
P[3]:=Point(3, 17);
P[4]:=Point(4, 17);
P[5]:=Point(5, 16);
P[6]:=Point(6, 16);
P[7]:=Point(7, 15);
P[8]:=Point(8, 15);
P[9]:=Point(9, 14);
P[10]:=Point(10, 14);
P[11]:=Point(11, 13);
P[12]:=Point(12, 12);
P[13]:=Point(13, 11);
P[14]:=Point(14, 9);
P[15]:=Point(15, 7);
P[16]:=Point(16, 5);
P[17]:=Point(17, 0);
result:=P;

```



end;

function GetRCPoints(Radius:Integer=3):TPoints;

begin

//通过给定圆角半径，计算出4分之1圆上各点坐标。

if Radius<2 then Radius:=2;

if Radius>18 then Radius:=18;

if (Radius mod 2)=0 then Radius:=Radius+1;

case Radius of

3:result:=RC3Radius;

5:result:=RC5Radius;

7:result:=RC7Radius;

9:result:=RC9Radius;

11:result:=RC11Radius;

13:result:=RC13Radius;

15:result:=RC15Radius;

17:result:=RC17Radius;

else

result:=RC3Radius;

end;//over case

end;

Procedure EditXY(VAR P:TPoints;x,y:integer;O:TPoint;xy:String='x');

var i,co:integer;

begin

//把P[i]的x乘以x,并加上o.x

XY:=UPPERCASE(XY);

co:=length(P)-1;

for i:=0 to co do

BEGIN

IF Pos('X',XY)<>0 THEN

P[I].X:=x*P[I].X;

IF Pos('Y',XY)<>0 THEN

P[I].Y:=y*P[I].Y;

P[I].X:=P[I].X+O.X;

P[I].Y:=P[I].Y+O.Y;

END;

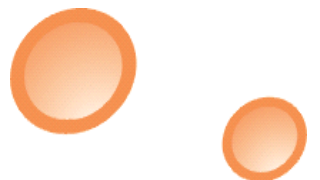
end;

Procedure draw2ColorRCRect(Canvas:TCanvas;A,C:TPoint;Radius:Integer=3;

Color1:TColor=clwhite;Color2:TColor=clblack;Visible:String='12345678');

var P2:TPoints; O:TPoint; R:Integer; C1:TColor;

begin



//在画布上画一个圆角矩形,A、B、C三个圆角是白色的,D角是黑色的。

```
C1:=Canvas. Pen. Color;
```

```
//
```

```
r:=Radius;
```

```
if r=0 then
```

```
begin
```

```
Draw2ColorRect(Canvas,A,C,Color1,Color2,Visible);
```

```
Exit;
```

```
end;
```

```
Canvas. Pen. Color:=Color1;
```

```
if Color1<>clnone then
```

```
Begin
```

```
//E
```

```
if pos('5',Visible)<>0 then
```

```
begin
```

```
P2:=GetRCPoints(Radius);
```

```
O:=Point(A.X+r,A.y+r);
```

```
EditXY(P2,-1,-1,0,'XY');
```

```
Canvas. Polyline(P2);
```

```
end;
```

```
//H
```

```
if pos('8',Visible)<>0 then
```

```
begin
```

```
P2:=GetRCPoints(Radius);
```

```
O:=Point(C.X-r,A.y+r);
```

```
EditXY(P2,-1,-1,0,'Y');
```

```
Canvas. Polyline(P2);
```

```
end;
```

```
//F
```

```
if pos('6',Visible)<>0 then
```

```
begin
```

```
P2:=GetRCPoints(Radius);
```

```
O:=Point(A.X+r,C.y-r);
```

```
EditXY(P2,-1,-1,0,'X');
```

```
Canvas. Polyline(P2);
```

```
end;
```

```
End;//if <>clnone
```

```
Canvas. Pen. Color:=Color2;
```

```
//G
```

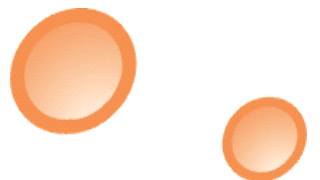
```
if Color2<>clnone then
```

```
Begin
```

```
if pos('7',Visible)<>0 then
```

```
begin
```

```
P2:=GetRCPoints(Radius);
```



```

O:=Point (C. X-r, C. y-r);
EditXY(P2, -1, -1, 0, ' ');
Canvas. Polyline (P2);
end;
End; //if <>clnone

Canvas. Pen. Color:=Color1;
if Color1<>clnone then
if pos('1', Visible)<>0 then
DrawLine(Canvas, Point (A. X, A. Y+R), Point (A. X, C. Y-R+1));

Canvas. Pen. Color:=Color2;
if Color2<>clnone then
Begin
if pos('2', Visible)<>0 then
DrawLine(Canvas, Point (A. X+R, C. Y), Point (C. X-R, C. Y));
if pos('3', Visible)<>0 then
DrawLine(Canvas, Point (C. X, C. Y-R), Point (C. X, A. Y+R-1));
End; //if <>clnone

Canvas. Pen. Color:=Color1;
if Color1<>clnone then
if pos('4', Visible)<>0 then
DrawLine(Canvas, Point (A. X+R, A. Y), Point (C. X-R, A. Y));
//
Setlength(P2, 0);
//
Canvas. Pen. Color:=C1;
end;

Procedure draw2Rectangle(Canvas:TCanvas;A, C:TPoint;Radius:Integer=3;
Color1:TColor=clwhite;Color2:TColor=clblack;Color3:TColor=clwhite;Color4:TColor
=clblack;
Visible:String='12345678');
begin
//画双重边框
draw2ColorRRect(Canvas, A, C, Radius, Color1, Color2, visible);
A:=Point (A. X+1, A. Y+1); C:=Point (C. X-1, C. Y-1);
draw2ColorRRect(Canvas, A, C, Radius, Color3, Color4, visible);
end;

```

然后按照下面的样子修改WMPaint()：

```

procedure TJHLButton.WMPaint(var Message: TWMPaint);

```



```

var .....
begin
.....
//3. 在b上画按钮表面的第三部分：边框
draw2Rectangle(b.Canvas, Point(0, 0), Point(width-1, height-1), 5,
clBtnHighlight, clBtnShadow, $00C08080, cl3DDkShadow);
//4. 在b上画按钮表面的第四部分：表示按钮拥有焦点的矩形虚线框
.....

```

看WMPaint()函数中的“//4. 在b上画按钮表面的第四部分：……”这一行，把下面的代码添加到这一行的下面：

```

if self.Focused then
begin
b.Canvas.Pen.Style:=psdot;
b.Canvas.Pen.Width:=1;
b.Canvas.Rectangle(4, 4, width-4, height-4);
end;

```

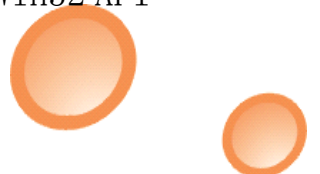
这6行代码的意思是：

如果按钮有焦点，那么在位图b上，画一个矩形虚线框

由于按钮边框的宽度是2，所以表示焦点的虚线框应该和边框之间保存距离。

五、按钮的圆角外形

圆角控件已经成为各大软件的流行风格，但Delphi6和Win32 API



却没有直接提供‘把矩形控件变成圆角控件的函数’，

十分不便于快速编程。

我曾经查阅了很多技术文章，但没有找到满意的解决方案，

最后还是自己亲手写了代码。

这至少耗去了我1个月的时间。

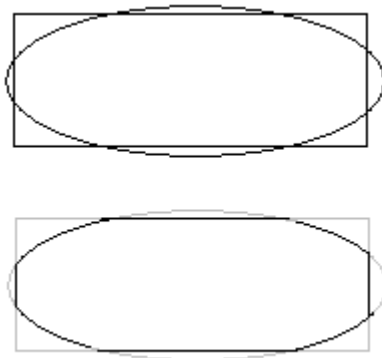
我的经验告诉我，圆角分成两种：正圆角、椭圆角，

国际软件中用的都是正圆角，而Delphi6和Win32 API提供的却是椭圆

角，采用椭圆角的风格，明显不如正圆角风格。

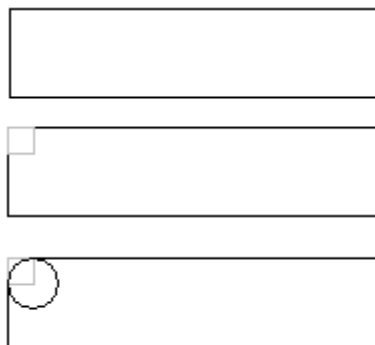
要想使自己的软件达到国际水准，所以非得自己攻破这项技术。

画椭圆角很简单，看看下面的图片就明白了：

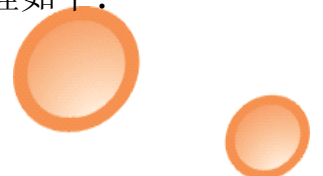


只要保留矩形和椭圆的公共部分，就能画出椭圆角矩形。

下面看看如何绘画正圆角矩形：



如上图所示，把一个矩形控件的左上角变成圆角的大致过程如下：



- 1、如果要做一个圆角大小等于3的圆角矩形，那么就先减去一个边长等于3的正方形；
- 2、然后以这个正方形的右下角为圆心作一个圆；
- 3、保留圆和矩形的公共部分，就得到了圆角。

看了这篇文章，大家就不需要亲手编写源代码，用现成的就行了。

我编写了一个函数，专门用来把按钮变成圆角：

```
Procedure RCControl (C:TWinControl;Radius:Integer=9;Visible:String='12345678');  
参数 'Radius' 用于指定圆角的大小，取值范围：0-19之间的奇数；  
参数visible用于指定把控件的哪个角变成圆角，一般不用自己设置，使用默认值就行了。  
'5' 代表矩形的左上角；'6' 代表矩形的左下角；  
'7' 代表矩形的右下角；'8' 代表矩形的右上角；
```

原理介绍完了，下面接着来编写我们的组件：TJHLButton

如何运用此函数，把按钮变成圆角的详细步骤如下：

0、在interface部分，添加如下代码：

```
.....  
  
interface  
  
Uses .....  
  
Type TPoints=Array of TPoint;  
  
type  
  
    TJHLButton = class(TButton)  
  
.....
```

1、找到“draw2Rectangle()”这个过程，然后在这个过程的后面添加下列函数：

```
function AddToPoints (Pl, P:TPoints):TPoints;
```



```

var i, co, il, col: integer;
r: tpoints;
begin
//把p1中的点添加到p
r:=p;
i:=length(r);
co:=length(p1);
setlength(r, i+co);
col:=i+co-1;
for il:=i to col do
r[il]:=p1[il-i];
result:=r;
end;

```

```

procedure AddPoints (var P1, P: TPoints);
VAR r: TPoints;
BEGIN
//把P1追加到P后面
r:=AddToPoints (p1, p);
p:=r;
END;

```

```

function AppendToPoints (P1: TPoint; P: TPoints): TPoints;
var i: integer;
r: tpoints;
begin
//把p1中的点添加到p
r:=p;
i:=length(r);
setlength(r, i+1);
r[i]:=p1;
result:=r;
end;

```

```

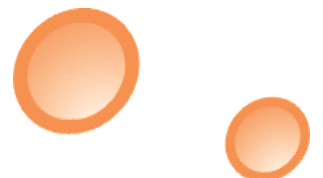
procedure AppendPoints (P1: TPoint; var P: TPoints);
VAR r: TPoints;
BEGIN
//把P1追加到P后面
r:=AppendToPoints (p1, p);
p:=r;
END;

```

```

Procedure ExChangeEach (Var P: TPoints);
var P1: TPoints; CO, I: Integer;

```

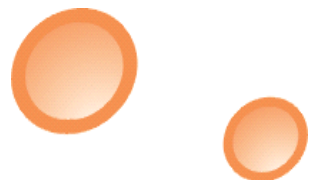


```

begin
//把P头尾交换
co:=Length(P);
SetLength(P1, co);
dec(co);
for i:=0 to co do
P1[CO-I]:=P[I];
P:=P1;
Setlength(P1, 0);
end;

Function
TheRCPoints(A, C:TPoint;Radius:Integer=3;Visible:String='12345678'):TPoints;
var P1,P2:TPoints; P,O:TPoint; R:Integer;
begin
//计算圆角矩形某个圆角上的点的坐标.
r:=Radius;
//E
if pos('5',Visible)<>0 then
begin
O:=Point(A.X+r, A.y+r);
P2:=GetRCPoints(Radius);
EditXY(P2, -1, -1, 0, 'XY');
AddPoints(P2, P1);
end else
begin
P:=A;
AppendPoints(P, P1);
end;
//F
if pos('6',Visible)<>0 then
begin
O:=Point(A.X+r, C.y-r-1);
P2:=GetRCPoints(Radius);
EditXY(P2, -1, -1, 0, 'X');
ExchangeEach(P2);
AddPoints(P2, P1);
end else
begin
P:=Point(A.X, C.Y);
AppendPoints(P, P1);
end;
//G
if pos('7',Visible)<>0 then

```




```

begin
O:=Point(C.X-r-1,C.y-r);
P2:=GetRCPoints(Radius);
EditXY(P2,-1,-1,0,'');
AddPoints(P2,P1);
end else
begin
P:=C;
AppendPoints(P,P1);
end;
//H
if pos('8',Visible)<>0 then
begin
O:=Point(C.X-r,A.y+r);
P2:=GetRCPoints(Radius);
EditXY(P2,-1,-1,0,'Y');
ExchangeEach(P2);
AddPoints(P2,P1);
end else
begin
P:=Point(C.X,A.Y);
AppendPoints(P,P1);
end;
result:=P1;
Setlength(P2,0);
end;

Function TheHRGN(A,C:TPoint;Radius:Integer=3;Visible:String='12345678'):HRGN;
var P1:TPoints; i,co:integer; P:Array [0..100] of TPoint;
Begin
//先计算圆角矩形上各个点，然后再次计算HRGN.
P1:=TheRCPoints(A,C,Radius,Visible);
co:=length(P1)-1;
for i:=0 to co do
p[i]:=P1[i];
Result:=CreatePolygonRGN(P,co+1,WINDING);
End;

Procedure RCControl(C:TWinControl;Radius:Integer=9;Visible:String='12345678');
var h:hrgn; P1,P2:TPoint;
begin
//把控件C变成圆角。左上角用5标示，左下角用6标示，右下角用7标示，右上角用8标示。
//若5678都在Visible中，那么控件的所有角都将变成圆角。
//若5678都不在Visible中，那么控件恢复矩形。

```



```

if c=nil then exit;
WITH C DO
BEGIN
P1:=Point(0,0); P2:=Point(WIDTH,HEIGHT);
END;
h:=TheHRGN(P1,P2,Radius,Visible);
setwindowrgn(c.Handle,h,true);
end;

```

2、在“Protected”部分插入一行：

```

procedure ConstrainedResize(var MinWidth, MinHeight, MaxWidth, MaxHeight:
Integer);override;

```

3、在“procedure Register;”的上面插入代码：

```

procedure TJHLButton.ConstrainedResize(var MinWidth, MinHeight, MaxWidth,
MaxHeight: Integer);
begin
inherited ConstrainedResize(MinWidth, MinHeight, MaxWidth, MaxHeight);
RCCControl(Self,3);
end;

```

为何要用到‘TPoints’这种类型呢？

由于要用到Win32 API函数‘*CreatePolygonRGN()*’，

所以需要计算圆角上点的坐标。

‘TPoints’类型的变量中可以保存大量的点。

大家无需太在意‘*RCCControl()*’的编程原理，只要懂得如何运用它就行了。

这个按钮组件TJHLButton，我已经做好了，作为Delphi的一个Demo，发布到了我的个人博客，只要通过Google搜索：DelphiDemos，就能找到了，或者直接搜索我的名字：金海龙，到我的博客里看看。欢迎大家免费下载：

其中包括：*.bpl,*.dpk,*.pas,*.dcu



本文的最后给出JHLButton.pas的完全源代码：

```
unit JHLButton;
```

```
interface
```

```
uses
```

```
    Windows, Messages, SysUtils, Classes, Controls, StdCtrls,  
Graphics;
```

```
Type TPoints=Array of TPoint;
```

```
type
```

```
    TJHLButton = class(TButton)
```

```
    private
```

```
        { Private declarations }
```

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

```
        procedure CMEnabledChanged(var Message: TMessage);  
message CM_ENABLEDCHANGED;
```

```
        procedure CMFontChanged(var Message: TMessage); message  
CM_FONTCHANGED;
```

```
    Protected
```



```
    { Protected declarations }

    procedure KeyDown(var Key: Word; Shift: TShiftState);
override;

    procedure KeyUp(var Key: Word; Shift: TShiftState);
override;

    procedure MouseDown(Button: TMouseButton; Shift:
TShiftState;
        X, Y: Integer); override;

    procedure MouseMove(Shift: TShiftState; X, Y: Integer);
override;

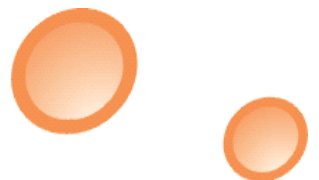
    procedure MouseUp(Button: TMouseButton; Shift:
TShiftState;
        X, Y: Integer); override;

    procedure ConstrainedResize(var MinWidth, MinHeight,
MaxWidth, MaxHeight: Integer);override;

public
    { Public declarations }

published
    { Published declarations }

end;
```



```
procedure Register;
```

```
implementation
```

```
procedure DrawLine(WhichCanvas:TCanvas;p1,p2:TPoint);
```

```
BEGIN
```

```
//从p1到p2画一条线段
```

```
with WhichCanvas do
```

```
begin
```

```
MoveTo(p1.x,p1.y);
```

```
LineTo(p2.x,p2.y);
```

```
end;
```

```
END;
```

```
Procedure
```

```
draw2ColorRect(Canvas:TCanvas;A,C:TPoint;Color1:TColor=clwh
```

```
ite;
```

```
Color2:TColor=clblack;Visible:String=' 1234');
```

```
VAR P1,P2:TPoint; l,t,r,b:Integer; C1:TColor;
```

```
Begin
```

```
//画两色矩形边框
```

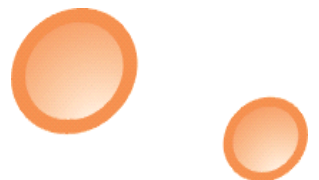
```
C1:=Canvas.Pen.Color;
```



```

//
l:=A.X; T:=A.Y; R:=C.X; B:=C.Y;
//
Canvas.Pen.Color:=Color1;
if Color1<>clnone then
BEGIN
if pos('l',Visible)<>0 then
begin
P1:=Point(l,t); P2:=Point(l,b);
drawline(Canvas,P1,P2);
end;
if pos('4',Visible)<>0 then
begin
P1:=Point(l,t); P2:=Point(r,t);
drawline(Canvas,P1,P2);
end;
END;
//-----
Canvas.Pen.Color:=Color2;
if Color2<>clnone then
BEGIN
if pos('2',Visible)<>0 then

```

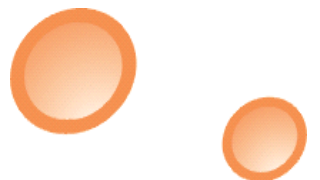


```
begin
P1:=Point(1, b); P2:=Point(r, b);
drawline(Canvas, P1, P2);
end;

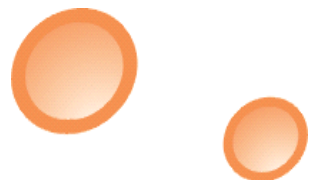
if pos('3', Visible) <> 0 then
begin
P1:=Point(r, b); P2:=Point(r, t-1);
drawline(Canvas, P1, P2);
end;
END;

//
Canvas.Pen.Color:=C1;
End;
```

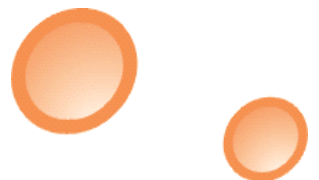
```
function RC3Radius:TPoints;
var P:TPoints;
begin
setlength(p, 4);
P[0]:=Point(0, 3);
P[1]:=Point(1, 3);
P[2]:=Point(2, 2);
```



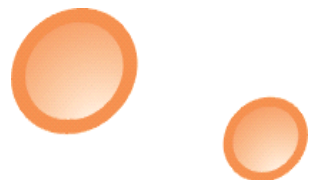
```
P[3]:=Point(3,0);  
  
result:=P;  
  
end;  
  
//end:30  
  
//begin:30  
  
function RC5Radius:TPoints;  
  
var P:TPoints;  
  
begin  
  
setlength(p,6);  
  
P[0]:=Point(0,5);  
P[1]:=Point(1,5);  
P[2]:=Point(2,5);  
P[3]:=Point(3,4);  
P[4]:=Point(4,3);  
P[5]:=Point(5,0);  
  
result:=P;  
  
end;  
  
//end:30  
  
//begin:30  
  
function RC7Radius:TPoints;  
  
var P:TPoints;  
  
begin
```




```
setlength(p, 8);  
P[0]:=Point(0, 7);  
P[1]:=Point(1, 7);  
P[2]:=Point(2, 7);  
P[3]:=Point(3, 6);  
P[4]:=Point(4, 6);  
P[5]:=Point(5, 5);  
P[6]:=Point(6, 3);  
P[7]:=Point(7, 0);  
result:=P;  
end;  
  
//end:30  
  
//begin:30  
  
function RC9Radius:TPoints;  
var P:TPoints;  
begin  
    setlength(p, 10);  
    P[0]:=Point(0, 9);  
    P[1]:=Point(1, 9);  
    P[2]:=Point(2, 9);  
    P[3]:=Point(3, 8);  
    P[4]:=Point(4, 8);
```



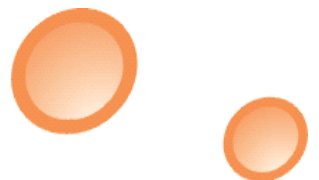
```
P[5]:=Point(5,7);
P[6]:=Point(6,7);
P[7]:=Point(7,5);
P[8]:=Point(8,3);
P[9]:=Point(9,0);
result:=P;
end;
//end:30
//begin:30
function RC11Radius:TPoints;
var P:TPoints;
begin
setlength(p,12);
P[0]:=Point(0,11);
P[1]:=Point(1,11);
P[2]:=Point(2,11);
P[3]:=Point(3,11);
P[4]:=Point(4,10);
P[5]:=Point(5,10);
P[6]:=Point(6,9);
P[7]:=Point(7,8);
P[8]:=Point(8,7);
```



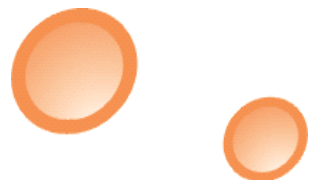
```
P[9]:=Point(9,6);  
P[10]:=Point(10,4);  
P[11]:=Point(11,0);  
result:=P;  
end;  
//end:30  
//begin:30  
function RC13Radius:TPoints;  
var P:TPoints;  
begin  
  setlength(p,14);  
  P[0]:=Point(0,13);  
  P[1]:=Point(1,13);  
  P[2]:=Point(2,13);  
  P[3]:=Point(3,13);  
  P[4]:=Point(4,12);  
  P[5]:=Point(5,12);  
  P[6]:=Point(6,12);  
  P[7]:=Point(7,11);  
  P[8]:=Point(8,10);  
  P[9]:=Point(9,9);  
  P[10]:=Point(10,8);
```



```
P[11]:=Point(11, 7);  
P[12]:=Point(12, 4);  
P[13]:=Point(13, 0);  
result:=P;  
end;  
//end:30  
//begin:30  
function RC15Radius:TPoints;  
var P:TPoints;  
begin  
  setlength(p, 16);  
  P[0]:=Point(0, 15);  
  P[1]:=Point(1, 15);  
  P[2]:=Point(2, 15);  
  P[3]:=Point(3, 15);  
  P[4]:=Point(4, 14);  
  P[5]:=Point(5, 14);  
  P[6]:=Point(6, 14);  
  P[7]:=Point(7, 13);  
  P[8]:=Point(8, 13);  
  P[9]:=Point(9, 12);  
  P[10]:=Point(10, 11);
```

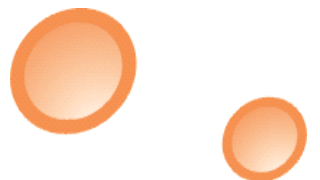


```
P[11]:=Point(11, 10);  
P[12]:=Point(12, 9);  
P[13]:=Point(13, 7);  
P[14]:=Point(14, 4);  
P[15]:=Point(15, 0);  
result:=P;  
end;  
  
//end:30  
  
//begin:30  
  
function RC17Radius:TPoints;  
var P:TPoints;  
begin  
  setlength(p, 18);  
  P[0]:=Point(0, 17);  
  P[1]:=Point(1, 17);  
  P[2]:=Point(2, 17);  
  P[3]:=Point(3, 17);  
  P[4]:=Point(4, 17);  
  P[5]:=Point(5, 16);  
  P[6]:=Point(6, 16);  
  P[7]:=Point(7, 15);  
  P[8]:=Point(8, 15);
```



```
P[9]:=Point(9,14);
P[10]:=Point(10,14);
P[11]:=Point(11,13);
P[12]:=Point(12,12);
P[13]:=Point(13,11);
P[14]:=Point(14,9);
P[15]:=Point(15,7);
P[16]:=Point(16,5);
P[17]:=Point(17,0);
result:=P;
end;
```

```
function GetRCPoints(Radius:Integer=3):TPoints;
begin
//通过给定圆角半径，计算出4分之1圆上各点坐标。
if Radius<2 then Radius:=2;
if Radius>18 then Radius:=18;
if (Radius mod 2)=0 then Radius:=Radius+1;
case Radius of
3:result:=RC3Radius;
5:result:=RC5Radius;
7:result:=RC7Radius;
```



```

9:result:=RC9Radius;

11:result:=RC11Radius;

13:result:=RC13Radius;

15:result:=RC15Radius;

17:result:=RC17Radius;

else

result:=RC3Radius;

end;//over case


end;

```

```

Procedure                               EditXY (VAR
P:TPoints;x,y:integer;O:TPoint;xy:String=' x' );
var i,co:integer;

begin

//把P[i]的x乘以x,并加上o.x

XY:=UPPERCASE (XY) ;

co:=length(P)-1;

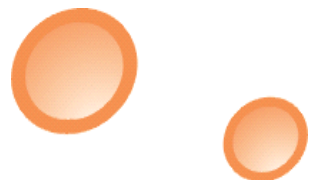
for i:=0 to co do

BEGIN

IF Pos (' X' ,XY)<>0 THEN

P[I].X:=x*P[I].X;

```



```
IF Pos('Y',XY)<>0 THEN
```

```
P[I].Y:=y*P[I].Y;
```

```
P[I].X:=P[I].X+O.X;
```

```
P[I].Y:=P[I].Y+O.Y;
```

```
END;
```

```
end;
```

```
Procedure
```

```
draw2ColorRCRect(Canvas:TCanvas;A,C:TPoint;Radius:Integer=3  
;
```

```
Color1:TColor=clwhite;Color2:TColor=clblack;Visible:String=  
'12345678');
```

```
var P2:TPoints; O:TPoint; R:Integer; C1:TColor;
```

```
begin
```

```
//在画布上画一个圆角矩形,A、B、C三个圆角是白色的,D角是黑色的。
```

```
C1:=Canvas.Pen.Color;
```

```
//
```

```
r:=Radius;
```

```
if r=0 then
```

```
begin
```

```
Draw2ColorRect(Canvas,A,C,Color1,Color2,Visible);
```

```
Exit;
```




```

end;

Canvas.Pen.Color:=Color1;

if Color1<>clnone then

Begin

//E

if pos(' 5',Visible)<>0 then

begin

P2:=GetRCPoints(Radius);

O:=Point(A.X+r,A.y+r);

EditXY(P2,-1,-1,0,'XY');

Canvas.Polyline(P2);

end;

//H

if pos(' 8',Visible)<>0 then

begin

P2:=GetRCPoints(Radius);

O:=Point(C.X-r,A.y+r);

EditXY(P2,-1,-1,0,'Y');

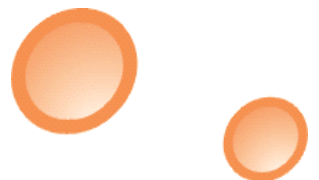
Canvas.Polyline(P2);

end;

//F

if pos(' 6',Visible)<>0 then

```



```

begin
P2:=GetRCPoints(Radius);
O:=Point(A.X+r,C.y-r);
EditXY(P2,-1,-1,0,'X');
Canvas.Polyline(P2);
end;
End;//if <>clnone
Canvas.Pen.Color:=Color2;
//G
if Color2<>clnone then
Begin
if pos('7',Visible)<>0 then
begin
P2:=GetRCPoints(Radius);
O:=Point(C.X-r,C.y-r);
EditXY(P2,-1,-1,0,'');
Canvas.Polyline(P2);
end;
End;//if <>clnone

Canvas.Pen.Color:=Color1;
if Color1<>clnone then

```



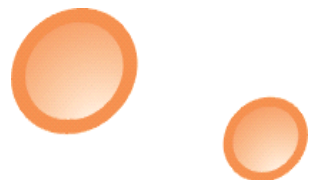
```

if pos('1',Visible)<>0 then
DrawLine(Canvas,Point(A.X,A.Y+R),Point(A.X,C.Y-R+1));

Canvas.Pen.Color:=Color2;
if Color2<>clnone then
Begin
if pos('2',Visible)<>0 then
DrawLine(Canvas,Point(A.X+R,C.Y),Point(C.X-R,C.Y));
if pos('3',Visible)<>0 then
DrawLine(Canvas,Point(C.X,C.Y-R),Point(C.X,A.Y+R-1));
End;//if <>clnone

Canvas.Pen.Color:=Color1;
if Color1<>clnone then
if pos('4',Visible)<>0 then
DrawLine(Canvas,Point(A.X+R,A.Y),Point(C.X-R,A.Y));
//
Setlength(P2,0);
//
Canvas.Pen.Color:=C1;
end;

```



Procedure

```
draw2Rectangle(Canvas:TCanvas;A,C:TPoint;Radius:Integer=3;  
Color1:TColor=clwhite;Color2:TColor=clblack;Color3:TColor=c  
lwhite;Color4:TColor=clblack;  
Visible:String=' 12345678' );  
begin  
//画双重边框  
if (Color1<>clnone)and(Color1<>clnone) then  
draw2ColorRCRect (Canvas,A,C,Radius,Color1,Color2,visible);  
A:=Point(A.X+1,A.Y+1); C:=Point(C.X-1,C.Y-1);  
if (Color3<>clnone)and(Color4<>clnone) then  
draw2ColorRCRect (Canvas,A,C,Radius,Color3,Color4,visible);  
end;
```

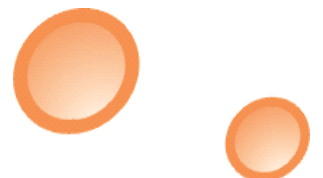
```
function AddToPoints(P1,P:TPoints):TPoints;  
var i,co,il,col:integer;  
r:tpoints;  
begin  
//把p1中的点添加到p  
r:=p;  
i:=length(r);  
co:=length(p1);
```



```
setlength(r, i+co);  
col:=i+co-1;  
for il:=i to col do  
  r[il]:=p1[il-i];  
result:=r;  
end;
```

```
procedure AddPoints(var P1,P:TPoints);  
VAR r:TPoints;  
BEGIN  
  //把P1追加到P后面  
  r:=AddToPoints(p1,p);  
  p:=r;  
END;
```

```
function AppendToPoints(P1:TPoint;P:TPoints):TPoints;  
var i:integer;  
r:tpoints;  
begin  
  //把p1中的点添加到p  
  r:=p;  
  i:=length(r);
```



```
setlength(r, i+1);
```

```
r[i]:=p1;
```

```
result:=r;
```

```
end;
```

```
procedure AppendPoints(P1:TPoint;var P:TPoints);
```

```
VAR r:TPoints;
```

```
BEGIN
```

```
//把P1追加到P后面
```

```
r:=AppendToPoints(p1, p);
```

```
p:=r;
```

```
END;
```

```
Procedure ExChangeEach(Var P:TPoints);
```

```
var P1:TPoints; CO, I:Integer;
```

```
begin
```

```
//把P头尾交换
```

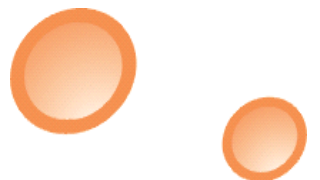
```
co:=Length(P);
```

```
SetLength(P1, co);
```

```
dec(co);
```

```
for i:=0 to co do
```

```
P1[CO-I]:=P[I];
```



```
P:=P1;
```

```
Setlength(P1, 0);
```

```
end;
```

```
Function
```

```
TheRCPoints(A, C:TPoint;Radius:Integer=3;Visible:String=' 123  
45678'):TPoints;
```

```
var P1,P2:TPoints; P, O:TPoint; R:Integer;
```

```
begin
```

```
//创建常用圆角矩形.
```

```
r:=Radius;
```

```
//E
```

```
if pos(' 5',Visible)<>0 then
```

```
begin
```

```
O:=Point(A.X+r, A.y+r);
```

```
P2:=GetRCPoints(Radius);
```

```
EditXY(P2, -1, -1, O, ' XY' );
```

```
AddPoints(P2, P1);
```

```
end else
```

```
begin
```

```
P:=A;
```

```
AppendPoints(P, P1);
```



```

end;

//F

if pos(' 6', Visible) <> 0 then

begin

O:=Point (A. X+r, C. y-r-1);

P2:=GetRCPoints(Radius);

EditXY(P2, -1, -1, 0, ' X' );

ExChangeEach(P2);

AddPoints(P2, P1);

end else

begin

P:=Point (A. X, C. Y);

AppendPoints(P, P1);

end;

//G

if pos(' 7', Visible) <> 0 then

begin

O:=Point (C. X-r-1, C. y-r);

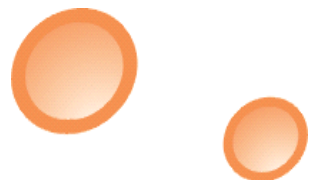
P2:=GetRCPoints(Radius);

EditXY(P2, -1, -1, 0, ' ' );

AddPoints(P2, P1);

end else

```



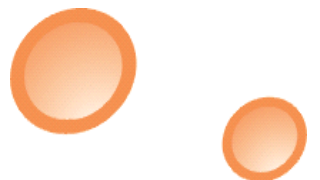

```

begin
P:=C;
AppendPoints (P, P1) ;
end;

//H
if pos('8',Visible)<>0 then
begin
O:=Point (C. X-r, A. y+r) ;
P2:=GetRCPoints (Radius) ;
EditXY (P2, -1, -1, 0, ' Y' ) ;
ExChangeEach (P2) ;
AddPoints (P2, P1) ;
end else
begin
P:=Point (C. X, A. Y) ;
AppendPoints (P, P1) ;
end;
result:=P1;
Setlength(P2, 0) ;
end;

```

Function



```

TheHRGN(A,C:TPoint;Radius:Integer=3;Visible:String='1234567
8'):HRGN;

var P1:TPoints; i,co:integer; P:Array [0..100] of TPoint;

Begin

//先计算圆角矩形上各个点，然后再次计算HRGN.

P1:=TheRCPoints(A,C,Radius,Visible);

co:=length(P1)-1;

//if co>100 then co:=100;

for i:=0 to co do

p[i]:=P1[I];

Result:=CreatePolygonRGN(P,CO+1,WINDING);

End;

```

Procedure

```

RCControl(C:TWinControl;Radius:Integer=9;Visible:String='12
345678');

var h:hrgn; P1,P2:TPoint;

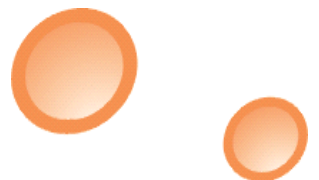
begin

//把控件C变成圆角。左上角用5标示，左下角用6标示，右下角用7
标示，右上角用8标示.

//若5678都在Visible中，那么控件的所有角都将变成圆角。

//若5678都不在Visible中，那么控件恢复矩形。

```



```
if c=nil then exit;

WITH C DO

BEGIN

P1:=Point(0,0); P2:=Point(WIDTH,HEIGHT);

END;

h:=TheHRGN(P1,P2,Radius,Visible);

setwindowrgn(c.Handle,h,true);

end;
```

```
function

CalTStringsHeight(Canvas:TCanvas;S:TStrings):Integer;

begin

//把s写到画布canvas上，若要垂直居中对齐，需要计算文字的高度,

这个函数用来计算S里文字的高度

Result:=S.Count*Canvas.TextHeight('金');

end;
```

```
function VAlignRct(R1:TRect;RctHeight:integer):TRect;

var i:integer; r:TRect;

begin

//计算居中对齐的区域
```



```

R:=r1;

i:=abs(R1.Bottom-R1.Top);

if RctHeight<i then

R:=Rect(r1.Left, r1.Top+round((i-RctHeight)/2), r1.Right, r1.Bottom);

result:=r;

end;

```

```

function

CalcTStringsWidth(Canvas:TCanvas;S:TStrings):Integer;

var i, co, ii, r:integer;

begin

//把s写到画布canvas上，若要垂直居中对齐，需要计算文字的高度，
这个函数用来计算S里文字的高度

co:=s.Count-1; r:=0;

for i:=0 to co do

begin

ii:=Canvas.TextWidth(s[i]);

if ii>r then r:=ii;

end; //over for

result:=r;

end;

```



```

function HAlignRct(R1:TRect;RctWidth:integer):TRect;
var i:integer; r:TRect;
begin
//计算居中对齐(水平)的区域
R:=r1;
i:=abs(R1.Right-R1.Left);
if RctWidth<i then
R:=Rect(r1.Left+round((i-RctWidth)/2),r1.Top,r1.Right,r1.Bottom);
result:=r;
end;

```

```

procedure
DrawButtonCaption(Canvas:TCanvas;R1:TRect;S:TStrings;TopAlign:boolean=false;
LeftAlign:boolean=true;Enabled:boolean=true;DisabledColor:TColor=clwhite);
var uFormat:cardinal; R2,R3:TRect; C:TColor;
begin

```

{左对齐写字符串, 多余的将被隐藏;

参数TopAlign指定: 在垂直方向是否靠上对齐



如果Enabled等于false，那么参数disabledcolor就会起作用，显示的文字将会带有阴影，阴影的颜色由disabledcolor决定。}

```
R2:=R1;

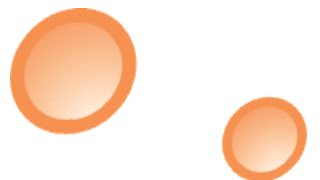
if topalign then
uFormat:=DT_LEFT or DT_Bottom
else
begin
uFormat:=DT_LEFT;
R2:=VAlignRect(R1, CalTStringsHeight(Canvas, S));
end;

if leftalign=false then
R2:=HAlignRect(R2, CalTStringsWidth(Canvas, S));

if not Enabled then
Begin
C:=Canvas.Font.Color;
Canvas.Font.Color:=DisabledColor;
R3:=Rect(R2.Left+1, R2.Top+1, R2.Right, R2.Bottom);
DrawText(Canvas.Handle, PChar(S.Text), -1, R3, uFormat);
Canvas.Font.Color:=C;
End;

DrawText(Canvas.Handle, PChar(S.Text), -1, R2, uFormat);//
```

这是Win32 API提供的函数



end;

Function GetRedFrom(C:TColor):byte;

begin

//读取C中的红色值

Result:=getRvalue(ColortoRGB(C));

end;

Function GetGreenFrom(C:TColor):byte;

begin

//读取C中的绿色值

Result:=getGvalue(ColortoRGB(C));

end;

Function GetBlueFrom(C:TColor):byte;

begin

//读取C中的蓝色值

Result:=getBvalue(ColortoRGB(C));

end;

procedure LimitRGB(var I:Integer);



begin

//i是red, green, blue三色中的一色，此函数使i不超过0-255的范围

if i>255 then i:=255;

if i<0 then i:=0;

end;

Function

AddRGBVal (C:TColor;I:Integer;ChangeRGB:String=' redgreenblue
'):TColor;

VAR r,g,b:integer; S:String;

begin

//累加C中的red, green, blue, 此函数使每个色值不超过0-255的范围

s:=lowercase(changergb);

r:=getredfrom(C); G:=getGREENfrom(C); B:=getBLUEfrom(C);

if pos(' red', S)<>0 THEN INC(R, I); LimitRGB(R);

if pos(' green', S)<>0 THEN INC(G, I); LimitRGB(G);

if pos(' blue', S)<>0 THEN INC(B, I); LimitRGB(B);

result:=rgb(r, g, b);

end;

Function

CaptionBarJB(BeginColor:TColor=clred;Num:integer=10;Step:in




```

teger=5;ChangeRGB:String='redgreenblue'):TBitmap;

var B:TBitmap; C1:TColor;

i,co:Integer;

begin

//标题栏渐变背景图

B:=TBitmap.Create; B.PixelFormat := pf24Bit;

B.Width:=1; B.Height:=num;

CO:=Num-1;

for i:=0 to co do

begin

c1:=AddRGBVal (BeginColor, i*abs(step), ChangeRGB);

if step<0 then

b.Canvas.Pixels[0, co-i]:=c1

else

b.Canvas.Pixels[0, i]:=c1;

end;

Result:=B;

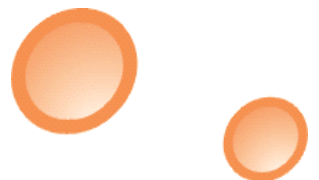
end;

Function

BtnJB(Color:TColor;BtnHeight:integer=25;Step:integer=3):TBitmap;

tmap;

```



```
begin  
result:=captionbarjb(Color,BtnHeight,Step);  
end;
```

```
procedure TJHLButton.CMEnabledChanged(var Message:  
TMessage);  
begin  
    inherited;  
    Invalidate;  
end;
```

```
procedure TJHLButton.CMFontChanged(var Message: TMessage);  
begin  
    inherited;  
    Invalidate;  
end;
```

```
procedure TJHLButton.KeyDown(var Key: Word; Shift:  
TShiftState);  
begin  
    inherited; if key=32 then Invalidate;  
end;
```



```
procedure TJHLButton.KeyUp(var Key: Word; Shift:
TShiftState);
begin
inherited; if key=32 then Invalidate;
end;
```

```
procedure TJHLButton.MouseDown(Button: TMouseButton; Shift:
TShiftState;
X, Y: Integer);
begin
inherited; Invalidate;
end;
```

```
procedure TJHLButton.MouseMove(Shift: TShiftState; X, Y:
Integer);
begin
inherited; if (ssLeft in Shift) or (ssRight in Shift) then
Invalidate;
end;
```

```
procedure TJHLButton.MouseUp(Button: TMouseButton; Shift:
```



```
TShiftState;
```

```
    X, Y: Integer);
```

```
begin
```

```
inherited; Invalidate;
```

```
end;
```

```
procedure TJHLButton.WMPaint(var Message: TWMPaint);
```

```
var b:TBitmap; c:TControlCanvas; cap:TStrings;
```

```
begin inherited;
```

```
//0. 准备一张要画到按钮表面的位图
```

```
b:=TBitmap.Create;
```

```
b.Width:=width; b.Height:=height;//这张图片的尺寸要和按钮的  
尺寸一样
```

```
//1. 在b上画按钮表面的第一部分：背景图
```

```
b.Canvas.StretchDraw(ClientRect, BtnJB(clbtnface, 25, -3)); //$  
00C08080
```

```
//2. 在b上画按钮表面的第二部分：文字
```

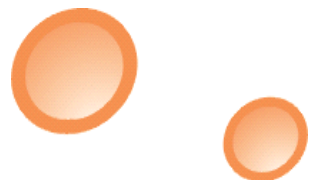
```
cap:=TStringlist.Create;
```

```
cap.Text:=caption;
```

```
b.Canvas.Brush.Style:=bsclear;
```



```
DrawButtonCaption(b.Canvas, ClientRect, cap,  
false, false, self.Enabled, clwhite);  
cap.Free;  
//3. 在b上画按钮表面的第三部分：边框  
draw2Rectangle(b.Canvas, Point(0, 0), Point(width-1, height-1),  
5,  
clbtnface, clBtnShadow, clwhite, clbtnface);  
//4. 在b上画按钮表面的第四部分：表示按钮拥有焦点的矩形虚线框  
if self.Focused then  
begin  
b.Canvas.Pen.Style:=psdot;  
b.Canvas.Pen.Width:=1;  
b.Canvas.Rectangle(4, 4, width-4, height-4);  
end;  
//5. 把b画到按钮的表面  
c:=TControlCanvas.Create;  
c.Control:=self;  
c.Draw(0, 0, b);  
c.Free;  
//6. 释放b所占用的内存空间  
b.Free;  
end;
```



```
procedure TJHLButton.ConstrainedResize(var MinWidth,
MinHeight, MaxWidth,
    MaxHeight: Integer);
begin
    inherited ConstrainedResize(MinWidth, MinHeight, MaxWidth,
MaxHeight);
    RControl(Self, 3);
end;

procedure Register;
begin
    RegisterComponents('DelphiDemos', [TJHLButton]);
end;

end.
```



{*****}

Delphi第三方组件信息

名称: TJHLButton

作者: 金海龙(软件工程师)

开发时间: 2011-1-11

类型: 按钮

简介: 这个按钮控件继承自TButton, 在按钮外观上做了很酷的改进, 拥有圆角边框和渐变背景图……

通过查看源代码, 个人开发者可以学习改进Delphi默认控件的方法, 非常适合广大中专院校的学生, 和那些旨在通过开发插件赚钱的个人开发者。

责任声明: 凡通过网络传播此文件的, 不得变更或删除作者信息。

相关内容: 如果了解所有源代码的含义或者编程原理, 可以通过Google搜索: “Delphi高级组件开发指南 第一篇”
我写了一篇文章, 专门讲解此组件的开发原理, 地址是:

<http://bigengineer.blog.163.com/blog/static/131780424201101115524181/>

如果要查阅更多相关编程资料, 请登录我的网站: <http://www.docin.com/delphi6x>

这个按钮组件中的关键技术: 把按钮变成圆角, 更改背景, TCanvas和多行文字……

Download :

<http://blog.ccidnet.com/blog.php?do=showone&uid=369088&type=blog&itemid=20070074>

*****}



