# Delphi 控件开发浅入深出

## Delphi 控件开发浅入深出(一)

有人说过“不会开发控件的 Delphi 程序员不是真正的程序员”。Delphi 正是由于高度的可扩展性和大量的第三方控件的支持才能吸引无数程序员挑剔的目光。即使是由于工作需要使用其他开发工具的开发者也常常怀念和 Delphi 度过的日日夜夜。接触 Delphi 已经一年多了，从当初对着 Delphi 组件面板上上百个控件不知所措，到现在已经可以根据需要开发一些有一定难度的控件，其中走过的路是十分艰辛的，所以特此写下这篇文章，将自己的经验留给后来者，也算是献给“同门师弟”的一份厚礼吧！

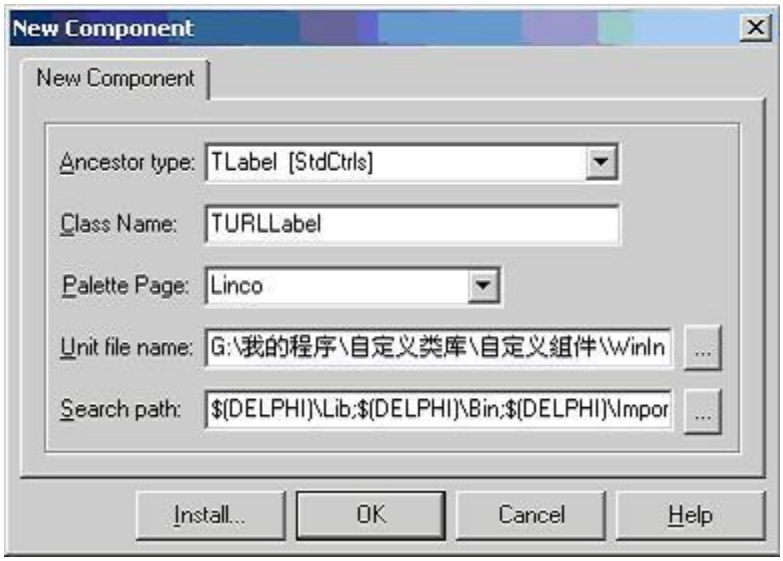
需要说明的一点是：在写这篇文章之前我假设读者已经对面向对象的基本知识有一定的了解，所以对于文章中面向对象相关的概念将不再展开讲述。

### 一、牛刀小试－TURLLabel 控件

我们从一个能够添加超链接的标签控件开始我们的控件开发之旅吧！

既然是 Label 我们就从 Tlabel 派生这个控件吧(其实从 TcustomLabel 派生最好，不过出于简单的目的我们这里先从 Tlabel 派生)！

1、选择“File”－>“New”－>“Component”,将弹出如下的对话框：




在 Ancestor type 中选择 Tlabel，ClassName 中填入 TURLLabel(名字可以任意取，但是要以 T 开头，否则的话会出现注册控件时候的问题)。这里 Ancestor type 代表控件的基类，TURLLabel 代表控件的名称。

2、超链接的视觉效果是带下划线的文字，所以我们覆盖父类的构造函数，在构造函数里修改控件的字体属性。

```
constructor TUrlLabel.Create(AOwner:TComponent);

begin

inherited Create(AOwner);

Cursor:=crHandPoint;

Font.Style:= [fsUnderline];

end;
```

代码解释：

（1）inherited Create(AOwner);这句的意思是执行父类的构造函数。我们制作控件的时候，如果覆盖了父类的构造函数，那么在新的构造函数中一定要首先调用父类的构造函

数，否则会引起错误。这是很多初学控件开发的人常常遇到的问题。当您把自己开发

的控件从面板上拖放到窗体时，如果跳出一个“Access Violent”的错误对话框的话，

那么十有八九是因为您忘了调用父类的构造函数。

（2）Cursor:=crHandPoint;

Font.Style:= [fsUnderline];

这三句的意思是修改标签的视觉效果。Cursor:=crHandPoint;是设定当鼠标移动到控件上时鼠标的形状为“手型”；Font.Style:= [fsUnderline]; 是设定文字的下划线效果。

3、既然是超链接控件，那么我们肯定要能使用户在使用控件的时候能在“Object Inspector”中对超链接的 URL 进行修改，所以我们应该为控件增加一个 Url 属性。

属性是访问控件字段的接口。通过属性，控件使用者可以间接读或者写控件的内部字段改变控件的状态。组件属性的声明需要以下几部分：属性名、属性类型、读方法（或读字段）、写方法（或写字段。如果没有写方法或写字段，则该属性为只读属性）。

属性在控件类声明的 Published 部分声明。在 Published 中声明的属性可以在设计期通过“Object Inspector”对属性值进行修改。如果声明在 Public 部分则不可以在设计期通过“Object Inspector”对属性值进行修改，但是可以在运行时通过代码进行读写。

在类声明的 Private 访问区域中添加如下字段声明：

FUrl: String;

在类声明中添加 Published 访问区域，并添加如下代码

property Url: String read FUrl write FUrl;

这段声明的意思是为控件添加一个 Url 属性，属性的类型是 string，在读 Url 属性时返回 Furl 的值，在写 Url 属性时设定 Furl 的值。

4、超链接的视觉效果有了，下面使它点击时调用浏览器打开 Url 指定的网址。

在 Delphi 控件的事件处理中很多事件都有对应的一个调度方法（这是设计模式中模板模式的典型应用）。比如在鼠标点击控件时，控件会首先调用 Click 方法，由 Click 方法进行相应的处理，而绝大多数调度方法都会引发一个事件句柄（关于事件句柄我们后边有深入的介绍）。比如 Tlabel 控件中在用户用点击 Label 时会首先调用控件的 Click 方法（被声明为 Protected 级别），Click 方法再触发 OnClick 事件。所以我们只要覆盖 Tlabel 的 Click 方法进行我们自己的处理就可以了。

在 Protected 部分添加如下的声明：

procedure Click;override;

在实现部分为 Click 方法写如下的代码：

procedure TUrlLabel.Click;

begin

ShellExecute(Application.Handle, nil, PChar(Url), nil, nil,SW_NORMAL);

inherited;

end;

代码解释：

（1）ShellExecute 的作用是用默认的程序打开第三个参数指定的文件。所以当第三个参数为一个 URL 时，则用浏览器打开这个网址。关于 ShellExecute 其他参数的使用方法

可以查阅 MSDN 或其他相关资料。

（2）Inherited;的作用是调用父类的 Click 方法来由父类来对鼠标单击事件做其他的处理。

5、源代码。

下面给出这个控件的全部源代码：

```
unit UrlLabel;
interface
uses
Windows, Messages, SysUtils, Classes, Controls, StdCtrls,Shellapi,
Graphics, Forms;
TUrlLabel = class(TLabel)
private
FUrl:AnsiString;
protected
procedure Click;override;
public
constructor Create(AOwner:TComponent);override;
published
property Url:AnsiString read FUrl write FUrl;
end;
procedure Register;
implementation
constructor TUrlLabel.Create(AOwner:TComponent);
begin
inherited Create(AOwner);
Cursor:=crHandPoint;
Font.Style:=[fsUnderline];
end;
procedure TUrlLabel.Click;
begin
ShellExecute(Application.Handle, nil, PChar(Url), nil, nil,SW_NORMAL); inherited;
procedure Register;
begin
```

RegisterComponents('Linco', [TUrlLabel]);//控件生成向导生成的注册控件用代码

end;

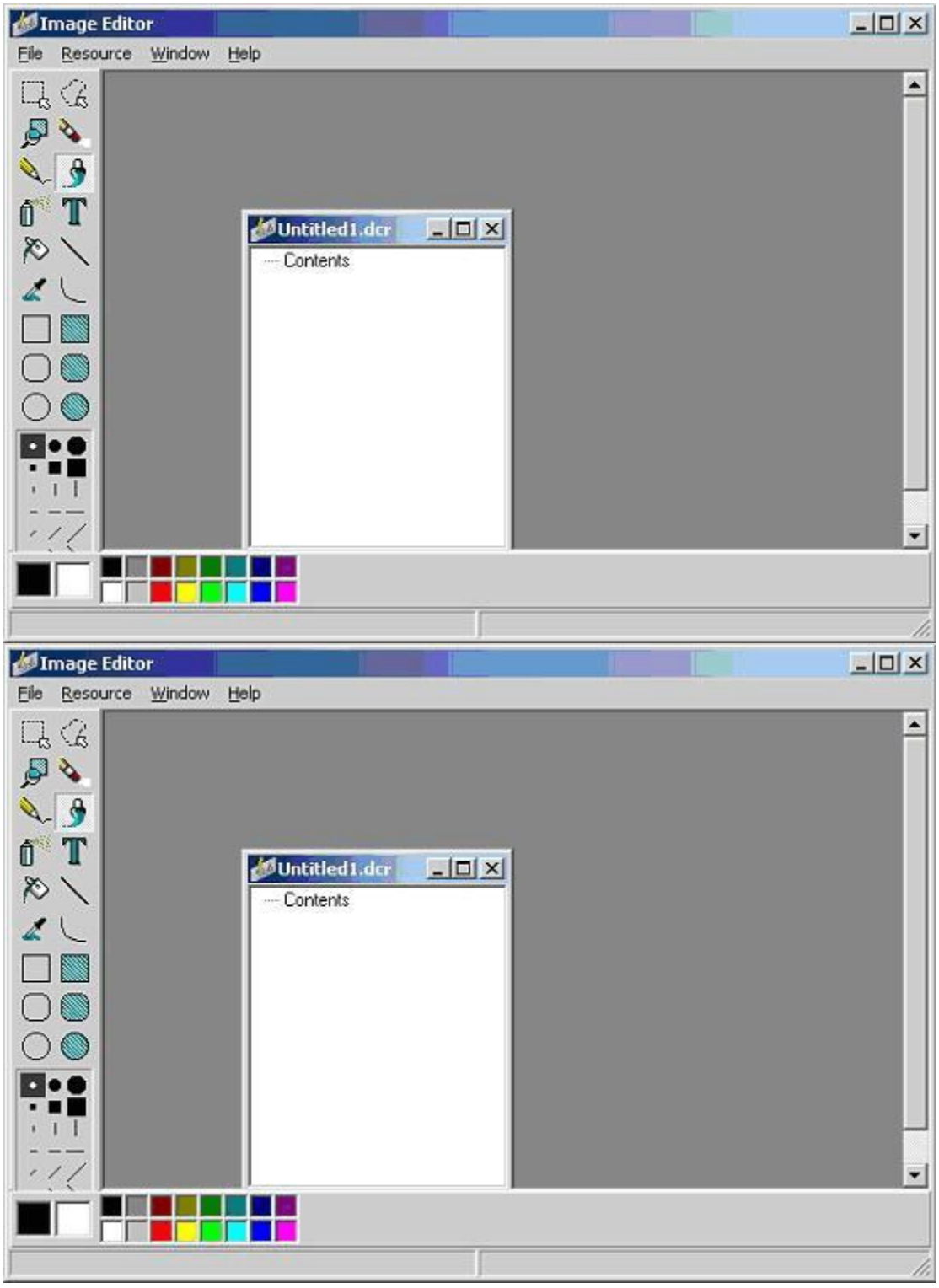
end.

6、为控件添加图标。

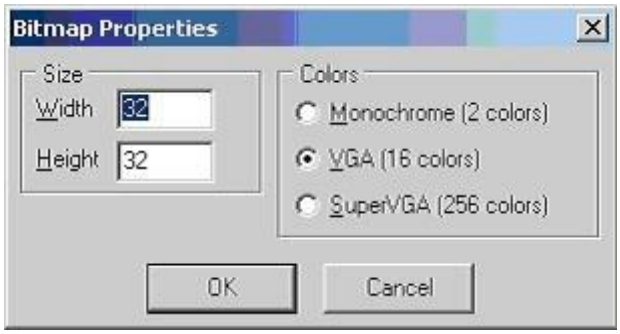
如果没有给自定义的控件定义图标，出现在控件面板上的自定义图标的图标是默认的图标，很没有“个性”，也不容易被用户与其他控件区别开来，所以我们需要给组件指定一个图标。

首先利用 Delphi 的 Image Editor 创建一个 24*24 的位图，并把它保存到一个 DCR 文件中。创建了一个位图后，就需要给位图命名了。位图的名称必须和控件的类名相同，且为大写，而 DCR 文件的名字则必须与控件所在单元的单元名相同。如我们上边定义的控件，位图的名字应该为 TURLLABEL，DCR 文件的名字应该是 UrlLabel.dcr，此 DCR 文件应该与组件的单元文件放在同一个目录下。

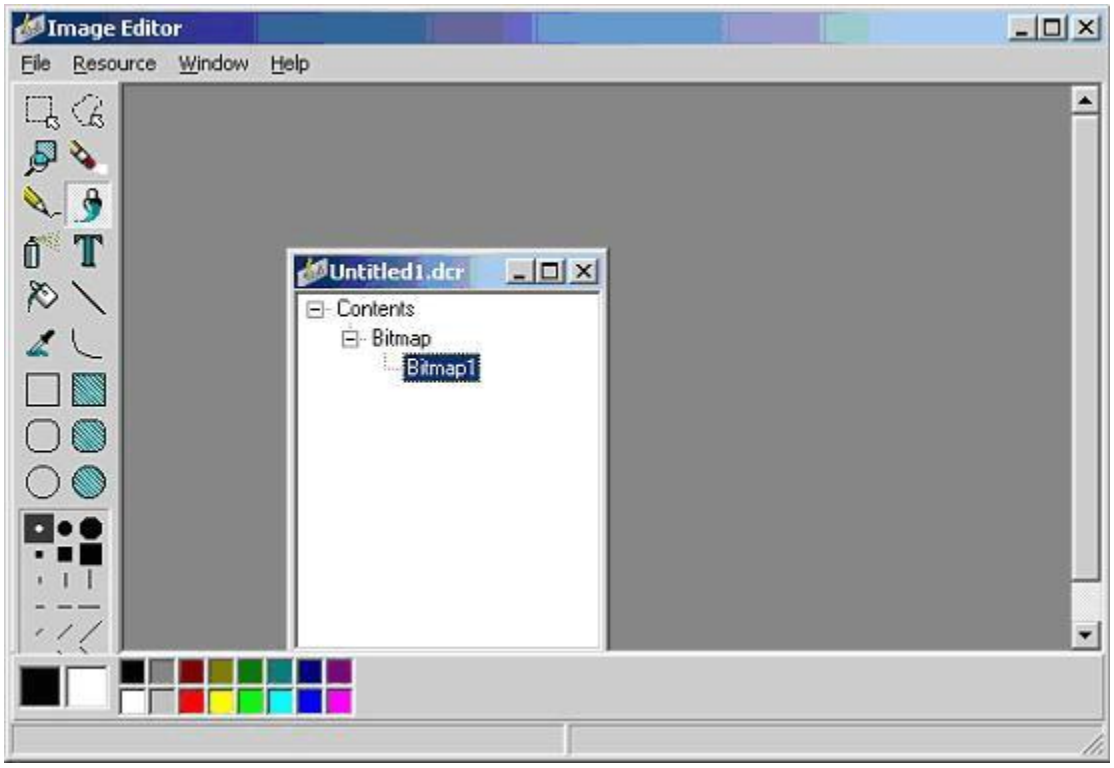
打开 Image Editor,选择”File”—>“New”—>“Component Resource File(.dcr)”，如下图：




在“Contents”上单击鼠标右键，选择“New”—>“BitMap”：




在 Width,Height 中都填入 24，点“OK”即可。




可以在 Bitmap1 上点右键选择“Rename”为位图重命名为 TURLLABEL，然后双击 TURLLABEL，就可以像使用“画图”一样为您的控件设计图标了。

7、注册组件。

点击 Componet—>Install Componet 进行自定义组件安装，此时将出现组件安装对话框。

在 Unit FilName 中输入控件单元文件的文件名（包括路径），点击“OK”,在弹出的 Package Editor 中按下 Install 按钮。如果安装成功系统就会提示安装成功。关闭 Package Editor 时，会提示您是否保存修改，点击 Yes 即可。

安装成功，建立一个测试程序。将 URLLabel 控件放到窗体上，设定 Url 属性为

http://www.sohu.com 运行程序，点击此 Label，就会弹出浏览器打开 http://www.sohu.com 这个网址。

思考题：

1、如何为控件添加一个图标？

2、Delphi 中的控件的共同基类是哪个类？

3、请做一个编辑框控件，当控件中输入的字符串是网址（以 http://开头）且用户在编辑框上

按回车时，用浏览器打开此网址。

**Delphi 控件开发浅入深出(二)**

## 二、控件开发纵览

通过开发上边这个控件，我们已经对 Delphi 控件开发有了基本的认识。下面我们将系统的讲述一下控件开发的知识。

制作控件第一件事就是选择适当的 Delphi 对象类型作为父对象，以派生新的对象。子对象可以继承父对象的全部非 private 部件，但不能摆脱不需要的部件。因此，所选父对象应尽可能多地包含子对象所需的属性、事件和方法，但不应包含子对象不需要的东西。Delphi 必须从 Tcomponent 或 Tcomponent 的子类派生。TComponent 是所有 Delphi 控件的基点，但若直接从 TComponent 派生新控件，很多东西就需要自己从头做起。一般只有非可视控件才直接从 TComponent 派生。Delphi 提供了若干专门用于制作控件（可视控件）的对象类型，都是从 TControl 和 TWinControl 派生而来。

TControl 的子类型用于非窗口式控件，TWinControl 的子类型则用于窗口式控件。除非特殊需要，一般不直接从 TControl 和 TWinControl 派生新控件，而是从其子类型派生。这样可以充分利用原有的属性、事件和方法，减少很多工作量。在这些控件类型中，非通用的属性、事件和方法都声明为 protected。这样可以禁止控件用户访问，又能被子类型继承和修改。在新控件中，可以简单地把继承来的属性和事件重新声明为 published，使控件用户能在设计期通过对象编辑窗口访问，也可以进而修改属性的默认值和读写方式，或是重载（override）事件处理子过程和其他控件方法，以修改其中的程序代码。重声明可以放宽访问权限，但不能相反，例如，不可能把 published 属性重声明为 private 或 protected。

Delphi 控件也是 Delphi 的类，所有的控件都有特定的结构。一般控件包括三大组成部分：属性、方法和事件，下面先介绍初学控件开发的最难懂的属性部分，其他部分我们将在以后章节为大家介绍。

属性主要部分就是属性的读写方法（或读写字段）。前面的例子用的是读写字段，也就是对属性的读写都通过对字段的读写来完成。下面为大家讲解一下读写方法的使用方法：

```
TmyComponent = class(TComponent)

Private

Fcount: Integer;

Procedure SetCount(Avalue: Integer);

Pulbished

Property Count: Integer read Fcount write SetCount;

End;
```

这个例子中当执行 MyComponent1.Count := 1;这样的代码时，将会导致 SetCount 方法执行，并且参数 Avalue 被指定为 1；当执行 I := MyComponent1.Count;方法时，会将 Fcount 的值返回给 I。

属性的声明语法允许属性声明的 Read 和 Write 部分用访问方法取代对象私有数据域。属性的读方法是不带参数的函数，返回同属性相同类型的值。通常读方法以 Get 开头。属性的写方法总是带一个参数的过程。写方法常常以 Set 开头。

**思考题：**

1、如何为控件添加属性？

2、从 TwinControl 类派生的控件的特点是什么？

**Delphi 控件开发浅入深出（三）**

三、开关控件 TlincoSwitch

用过 Delphi1（好古老的东东呀！）的人相信都记得这个开关控件，不知道当初 Borland 为什么把这么一个在开发普通应用程序中应用不到的工控控件放到 Delphi 中，而且在 Delphi2 及其以后的版本中再也没有见过它的身影。让我们怀着怀旧的心情把这位“开国元老”请出来吧！

1、建立位图资源文件：

用 Image Editor 建立一个 Res 文件，并在文件中分别建立下面两个位图，并分别命名为 SWITCHON、SWITCHOFF。保存此 Res 到控件单元所在目录下。

2、写控件代码。

```
unit LincoSwitch;

interface

uses

SysUtils, Classes, Controls, Graphics, Windows;

type

TLincoSwitch = class(TCustomControl)

private

FIsOn: Boolean;

FPicOn: Graphics.TBitmap;

FPicOff: Graphics.TBitmap;

procedure FSetIsOn(AValue: Boolean);

protected

procedure Click;override;

procedure Paint;override;

public

constructor Create(AOwner: TComponent);override;

destructor Destroy;override;

published

property IsOn: Boolean read FIsOn write FSetIsOn;
```

```
property OnClick;
property OnKeyDown;
property OnKeyPress;
property OnKeyUp;
property OnCanResize;
property OnDblClick;
property OnMouseDown;
property OnMouseMove;
property OnMouseUp;
property OnMouseWheel;
property OnResize;
end;
procedure Register;
implementation
{$R *.res}
procedure LoadBitmapFromRes(ABitmapId: string; ABitmap: Graphics.TBitmap);
begin
ABitmap.LoadFromResourceName(hInstance, ABitmapId);//从资源文件中读取位图 end;
constructor TLincoSwitch.Create(AOwner: TComponent);
begin
inherited Create(AOwner);
FPicOn := Graphics.TBitmap.Create;
FPicOff := Graphics.TBitmap.Create;
LoadBitmapFromRes('SWITCHON', FPicOn);
LoadBitmapFromRes('SWITCHOFF', FPicOff);
Invalidate;
end;
destructor TLincoSwitch.Destroy;
```

```
begin
FPicOn.Free;
FPicOff.Free;
inherited;
end;
procedure TLincoSwitch.Click;
begin
IsOn := not IsOn;//改变按钮的状态
Invalidate;
inherited;
end;
procedure TLincoSwitch.Paint;
begin
//画开关图案
if IsOn then
StretchBlt(Canvas.Handle, 0, 0, self.Width, self.Height,
FPicOn.Canvas.Handle,
0, 0, FPicOn.Width, FPicOn.Height,SRCCOPY)
else
StretchBlt(Canvas.Handle, 0, 0, self.Width, self.Height,
FPicOff.Canvas.Handle,
0, 0, FPicOff.Width, FPicOff.Height,SRCCOPY);
end;
procedure TLincoSwitch.FSetIsOn(AValue: Boolean);
begin
FIson := AValue;
Invalidate;
end;
```

procedure Register;

begin

RegisterComponents('Linco', [TLincoSwitch]);

end;

end.

3、代码分析

（1）、因为我们要在控件表面上将按钮的图案画出来，所以我们选择 TcustomControl

做为父类控件，因为它有个 Canvas 属性，我们可以利用 Canvas 在控件表面作图。不选用 Tcontrol 的原因是因为它有很多我们不需要的属性。

（2）、ABitmap.LoadFromResourceName(hInstance, ABitmapId);是从资源文件中读取 Id 为 AbitmapId 的位图，关于资源文件的使用请参考其他相关资料。注意代码中的“{$R *.res}”，它的作用是将资源文件编译到程序文件中，如果没有这个预编译条件，程序将会出现错误。

（3）、StretchBlt 是将位图画到画板上，使用方法请参考 MSDN。

（4）、我们为控件增加了 IsOn 属性。这个布尔属性用来表示开关的状态（开/关）。

从 property IsOn: Boolean read FIsOn write FSetIsOn;我们可以看出这个属性是个可读可写的属性。当读这个属性时会将 FisOn 的值返回给调用者，而写属性时则会调用 FsetIsOn 方法，并将赋给属性的值做为参数传递给 FsetIsOn。在 FsetIsOn 方法中，有如下实现代码： FIson := AValue;

Invalidate;

首先将 Fison 设置为参数传递来的值，然后调用 Invalidate;要求重画控件，以告诉用户控件的状态已经改变，这一点是使用写字段无法做到的。

（5）

FPicOn: Graphics.TBitmap;

FPicOff: Graphics.TBitmap;

是声明两个.Tbitmap 类型变量以保存控件的开关两种状态的图案。

（6）

procedure Click;override;

procedure Paint;override;

分别是覆盖父类中相应的调度方法。当控件被鼠标单击时，Click 方法会被调用，我们将在 Click 中改变控件的开关状态；Paint 方法则在用户调用 Invalidate 方法或控件发生重画时调用，我们一般在这个方法绘制控件的图案。

（7）、TcustomControl 中又很多事件处理句柄。比如 OnClick、OnKeyDown 等，但是它把他们声明成了 Protected 保护级别，所以我们在 Object Inspector 中看不到他们，如果我们要他们可以在 Object Inspector 中被用户编辑的话，只要在 Published 中重新声明他们即可，不用写他们的读写方法，只要使用：Property 属性名；

这样的方法就可以。比如这个例子中的：Property Onclick;

**思考题：**

1、做一个有特效的按钮控件，当鼠标按下时按钮是一个红色边框的空心圆，当鼠标松开时按钮是一个淡绿色边框的空心圆。

2、对于父类控件中为 protected 的属性，如果想将它在子类控件中公布，应该怎么做？请思考 Delphi 为什么要将一些属性设为 protected 级别？

**Delphi 控件开发浅入深出（四）**

**四、对特定字符串敏感的 Edit 控件**

我们这个控件将演示控件的自定义事件的书写。这个控件有一个类型为 string 的 SensitiveText 属性，当用户在输入框中输入的文字为 InvalidText 时就会触发

OnSensitiveText 事件。按照惯例，我先把源码展示给大家：

```
unit TextSenseEdit;

interface

uses

    SysUtils, Classes, Controls, StdCtrls;

type

  TSensitiveTextEvent = procedure(AText: string) of object;//方法指针

  TTextSenseEdit = class(TEdit)

  private

    FSensitiveText: string;

    FOnSensitiveText: TSensitiveTextEvent;

    procedure SetSensitiveText(AValue: string);

  protected

    procedure Change;override;

  public

  published

    property SensitiveText: string read FSensitiveText write SetSensitiveText;

    property OnSensitiveText: TSensitiveTextEvent read FOnSensitiveText write
  FOnSensitiveText;

end;

procedure Register;

implementation

procedure Register;

begin
```

```
        RegisterComponents('Linco', [TTextSenseEdit]);
end;
procedure TTextSenseEdit.Change;
begin
        inherited;
        if Text = SensitiveText then
                if Assigned(OnSensitiveText) then
                        OnSensitiveText(Text);
end;
procedure TTextSenseEdit.SetSensitiveText(AValue: string);
begin
        FSensitiveText := AValue;
end;
end.
```

代码解释：

（1）、SensitiveText 属性的添加方法大家已经熟悉了，这里不多解释。

（2）、正如大家猜测的，Change 方法正是编辑框文字发生变化时的调度方法，它将引起 OnChange 事件。我们可以在这个方法中监控编辑框文字发生的变化，当文字等于 SensitiveText 就触发 OnSensitiveText 事件（具体的实现方法在后边解释）。

（3）、Delphi 中的控件的事件机制是通过方法指针来实现的。声明方法指针的格式为：方法指针名称 = procedure(参数列表) of object;

声明事件属性的方法与声明普通属性的方法相同。在我们这个例子中，我们首先声明一个 FOnSensitiveText: TSensitiveTextEvent;私有变量，然后 property OnSensitiveText: TSensitiveTextEvent read FOnSensitiveText write FOnSensitiveText; 声明事件属性。这样注册控件后，当用户把控件放到窗体中后，就会在 Object Inspector 中 Evnets 页中出现 OnSensitiveText 事件，我们就可以像使用其他事件一样使用这个事件了。

但是我们现在只是声明了一个事件属性，并没有书写任何代码来激发这个事件。我们应该在合适的时候激发此事件，显而易见我们应该在 Change 方法中激发此事件：procedure TTextSenseEdit.Change;

```
begin
inherited;
if Text = SensitiveText then
if Assigned(OnSensitiveText) then
OnSensitiveText(Text);
end;
```

当 if Text = SensitiveText 时就判断控件使用者是否为 OnSetSensitiveText 写代码了（准确的说是是否为 OnSetSensitiveText 事件句柄赋值了），如果写代码了则调用 OnSetSensitiveText(Text);来激发 OnSetSensitiveText 事件，并把控件的 Text 传递给方法的 Avalue 参数。正如“方法指针”这个名字一样，被声明为方法指针类型的变量可以当作方法使用，用来激发事件。VCL 已经为我们预定义了一些常用的事件句柄，我们直接拿来使用：TnotifyEvent，TmouseEvent，TmouseMoveEvent，TkeyPressEvent 等，具体可以参考 VCL 源码。

**思考题：**

1、做一个支持累加运算的文本编辑框控件，用户可以在编辑框中输入正整数。当用户按回车时，如果编辑框中输入的不是正整数（为负数、小数或一般字符串）则触发控件的 OnError 事件；如果输入的是正整数，则开始计算从 1 到用户输入的那个正整数中所有整数的和（用 1＋2＋3+……这种累加的办法实现，不要用 (1+n)*n/2 这种直接计算的方法），并且在计算工程中如果发现计算的中间结果位数是 5，则触发 OnTailFive 事件。

**Delphi 控件开发浅入深出（五）**

五、复合控件

复合控件是 Delphi 控件中非常重要的一种控件，复合控件就是将两个或两个以上的控件重新组合成一个新的控件。例如 TspinEdit、TlabeledEdit、TDBNavigator 等就是复合控件，TDBNavigator 其实就是在一个 Panel 放上若干个 Button 而已。制作一个复合控件时，我们一般从 TwinControl 派生控件。

我们这次做的控件是拥有一个 Edit 编辑框和一个 Button 按钮的复合控件，在用户在编辑框中输入文字的过程中，Button 将随时显示编辑框中文字的长度。我们把控件的源码先展示给大家。

```
unit EditButton;

interface

uses

SysUtils, Classes, Controls, StdCtrls, Messages;

type

TEditButton = class(TWinControl)

private

FEdit: TEdit;

FButton: TButton;

FText: string;

procedure FSetText(AValue: string);

procedure OnEditChange(Sender: TObject);

protected

procedure WMSize(var Msg: TMessage);message WM_SIZE;

public

constructor Create(AOwner: TComponent);override;

destructor Destroy;override;

published

property Text: string read FText write FSetText;

end;
```

```
procedure Register;
implementation
procedure Register;
begin
RegisterComponents('Linco', [TEditButton]);
constructor TEditButton.Create(AOwner: TComponent); begin
inherited;
FEdit := TEdit.Create(nil);
FEdit.Parent := self;
FEdit.Top := 0;
FEdit.Left := 0;
FEdit.Height := Height;
FEdit.Width := Width div 2;
FEdit.OnChange := OnEditChange;
FButton := TButton.Create(nil);
FButton.Parent := self;
FButton.Top := 0;
FButton.Left := Width div 2;
FButton.Height := Height;
FButton.Width := Width div 2;
end;
destructor TEditButton.Destroy;
begin
FEdit.Free;
FButton.Free;
inherited;
end;
procedure TEditButton.FSetText(AValue: string); begin
```

FEdit.Text := AValue;

end;

procedure TEditButton.OnEditChange(Sender: TObject); begin

FButton.Caption := IntToStr(Length(FEdit.Text)); end;

procedure TEditButton.WMSize(var Msg: TMessage); begin

FEdit.Height := Height;

FEdit.Width := Width div 2;

FButton.Left := Width div 2;

FButton.Height := Height;

FButton.Width := Width div 2;

end.

代码解释：

（1）、我们首先定义了两个变量

FEdit: TEdit;

FButton: TButton;

分别代表复合控件中的文字编辑框和按钮。

（2）所谓复合控件说简单一点就是在一个共同的基板上将组成复合控件的各个控件（可以叫做子控件）画出来。所以我们在构造函数中建立各个子控件，然后分别设定它们的位置等属性。

以文字编辑框为例：

FEdit := TEdit.Create(nil);

的作用是建立编辑框控件。如果 Create 的参数指定为 nil,则子控件在设计状态是可以响应用户的操作的；而如果设定为 self(即设定子控件的父控件为基板)，则子控件在设计时时不可响应用户操作的，如果设定为 self 则析构函数中就不用 Fedit.Free 来销毁对象了，对象会自动销毁。

FEdit.Parent := self;的作用是设定子控件的父控件，如果没有这一句则控件是无法显示的。

FEdit.Top := 0;

FEdit.Left := 0;

FEdit.Height := Height;

FEdit.Width := Width div 2;

这四句是设定控件在基板上的相对位置的，这里的 Top,Left 不是相对于窗体的，而是相对于基板的。

FEdit.OnChange := OnEditChange;

则是设定编辑框控件的 OnChange(文字改变事件)的处理句柄为 OnEditChange；

（1）用户有可能在设计时或运行时通过代码改变控件的大小，这时控件中子控件的顺序就会变得乱七八糟，所以需要相应控件的 WM_SIZE 事件（控件大小发生变化的事件）重新设定子控件的位置，大小等。函数 WMSize 的作用就是这样的。

安装控件后发现控件已经可以正确运行了，但是还有一个问题，就是这个控件没有了 Onclick,Onchange 等必须的属性。我们只要为控件增加事件处理句柄属性，然后把事件处理句柄属性的读写方法都指向子控件的事件处理句柄属性即可。例如我们为控件增加 OnClick 事件，这个事件发生在用户单击按钮时，我么只要在 Pulished 部分增加如下代码：property OnClick: TnotifyEvent read GetOnClick write SetOnClick

在 Private 中增加如下方法声明：

function GetOnclick: TnotifyEvent;

procedure SetOnclick(AValue: TnotifyEvent);

这两个方法的实现分别为：

function TeditButton. GetOnclick: TnotifyEvent;

begin

result := Fbutton.Onclick;

end;

procedure TeditButton. SetOnclick(AValue: TnotifyEvent);

begin

Fbutton.OnClick := Avalue;

end;

思考题：

1、做一个模仿播放器中的操作按钮的复合控件，控件由三个按钮组成，分别是“播放”、“暂停”、“停止”，请按照正常的逻辑关系，处理这三个按钮的可用/不可用关系。（提示：可以参考 TDBNavigator 的源代码）

**Delphi 控件开发浅入深出（六）**

六、控件手拉手——控件关联的实现

控件的关联在 Delphi 中也是很常见的，我们可以设定一个控件的某个属性指向另一个控件。比如我们在窗体上放上 Tedit,TpopupMenu 两个控件，然后设定 Tedit 的 PopupMenu 属性为 TpopupMenu 控件，运行后在 Tedit 点击右键就会弹出刚才设定的那个 TpopupMenu

菜单，也就是说 Tedit,TpopupMenu 联手完成了任务。再比如 TDBEdit 控件的 DataSource

属性就可以指向一个 TdataSource 控件，这样就可以在 TDBEdit 控件中显示 TdataSource

输出的某个字段的值了。

下面我们将写一个简单的实现控件关联的控件。这个控件派生于 Tedit,它可以与一个 Tlabel 控件关联，在控件的编辑框中输入文字时，与它关联的 Tlabel 控件的文字将随着它而变化。代码如下：

```
unit MyEdit;

interface

uses

SysUtils, Classes, Controls, StdCtrls;

type

TMyEdit = class(TEdit)

private

FLinkLabel: TLabel;

procedure FSetLinkLabel(AValue: TLabel);

protected

procedure Notification(AComponent: TComponent;Operation: TOperation);

override;

procedure Change;override;

public

published

property LinkLabel: TLabel read FLinkLabel write FSetLinkLabel;
```

```
end;
procedure Register;
implementation
procedure Register;
begin
RegisterComponents('Linco', [TMyEdit]);
end;
procedure TMyEdit.Change;
begin
inherited;
if LinkLabel <> nil then
LinkLabel.Caption := Text;
end;
procedure TMyEdit.FSetLinkLabel(AValue: TLabel);
begin
FLinkLabel := AValue;
if AValue <> nil then
FLinkLabel.FreeNotification(self);
end;
procedure TMyEdit.Notification(AComponent: TComponent;
Operation: TOperation);
begin
inherited;
if (Operation = opRemove) and (AComponent = LinkLabel) then
LinkLabel := nil;
end;
end.
```

代码解释：

（1）、我们只要将控件的任意一个属性的类型设定为另外一个控件的类名称，那么我们就可以在控件的 Object Inspector 中将这个属性指向那个控件（或那个控件的派生控件）的一个实例。比如本例中我们增加了 LinkLabel 属性，它的类型为 Tlabel，所以我们就可以把 LinkLabel 属性指向一个标签控件。

（2）、请注意 FsetLinkLabel 中的这段代码：

```
if AValue <> nil then

FLinkLabel.FreeNotification(self);
```

如果我们将控件关联属性指向了一个控件，可是后来又将被指向的控件删除了，那么我们的控件关联属性是不会自动删除的，这样就会造成控件关联属性指向的控件不存在的现象。我们必须自动感知被关联控件的删除并重新设定控件关联属性为不指向任何控件，这样就避免了错误的发生。

FLinkLabel.FreeNotification(self);的作用就是这样的。它调用控件的 FreeNotification 方法（在 Tcomponent 中定义）向被指向的控件注册一个“消息”，当被指向控件被删除时，会向所有向他注册的控件发送一个它被删除的消息，此时向他注册的控件就会触发 Notification 方法，这样我们就可以自动感知被指向控件的状态了。这是设计模式中 Observer(观察者)模式的典型应用。

既然向他注册的控件就会触发 Notification 方法，我们就覆盖父类的 Notification 方法，写出如下的代码：

```
if (Operation = opRemove) and (AComponent = LinkLabel) then

LinkLabel := nil;
```

这句话的意思是：如果控件被删除并且被删除的控件（因为我们的控件可能向多个控件注册了消息）是 LinkLabel，那么我们就设定 LinkLabel 属性不指向任何控件。

（3）覆盖父类的 Change 调度方法。在此方法里为连接的 LinkLabel 的 Caption 赋值就达到我们的目的了。

思考题：

1、做一个 Label 控件，给它增加一个 DataSource 属性，该属性可以指向一个 TdataSource 类型的控件，它有一个 GetRecordCount 方法。当调用此方法时，就在 Label 控件中显示这个 DataSource 对应的数据集中的记录的条数。

**Delphi 控件开发浅入深出（七）**

**对话框控件的制作**

Delphi 中有很多对话框组件，例如 TopenDialog、TfontDialog 等。这些控件的特点就是虽然是不可视控件，但是在运行时都有一个可视化的效果，比如 TopenDialog 的可视化效果就是一个打开对话框。我们这次将开发一个日期对话框控件，当我们调用控件的 Execute 方法（不一定非要使用 Execute 方法，不过大部分对话框控件都是使用这个方法，我们也就按照惯例来了）时，就会弹出一个可以选择日期的对话框，我们选择一个日期后，点击“确定”则 Execute 返回 True,点击“取消”则 Execute 返回 False。我们可以读取 Date 属性来得到用户选择的日期，也可以修改此属性来改变对话框的初始日期。

1、新建一个对话框。在对话框窗体上放置一个 TmonthCalendar 组件，命名为 Cal,窗体名称改为 FormDate。在窗体上放置两个按钮，一个按钮的 Caption 为“确定(&O)”，ModalResult 为 mrOk，一个按钮的 Caption 为“取消(&C)”，ModalResult 为 mrCancel。设计好的窗体如下图所示：


2、为窗体添加两个 Public 访问级的方法：

function GetSelDate: TDate;

procedure SetInitDate(AValue: TDate);

代码如下：

function TFormDate.GetSelDate: TDate;

begin

result := cal.Date;

end;

procedure TFormDate.SetInitDate(AValue: TDate); begin

```
cal.Date := AValue;
end;
```

3、新建一个控件，派生自 Tcomponent。

代码如下：

```
unit DateDialog;
interface
uses
SysUtils, Classes, Controls, frmDlg;
type
TDateDialog = class(TComponent)
private
FDlg: TFormDate;
function GetDate: TDate;
procedure SetDate(AValue: TDate);
protected
public
constructor Create(AOwner: TComponent);override; function Execute: Boolean;
published
property Date: TDate read GetDate write SetDate; end;
procedure Register;
implementation
procedure Register;
begin
RegisterComponents('Linco', [TDateDialog]);
end;
constructor TDateDialog.Create(AOwner: TComponent); begin
inherited Create(AOwner);
FDlg := TFormDate.Create(self);
```

```
end;
function TDateDialog.Execute: Boolean;
begin
result := (FDlg.ShowModal = mrOK);
end;
function TDateDialog.GetDate: TDate;
begin
result := FDlg.GetSelDate;
end;
procedure TDateDialog.SetDate(AValue: TDate);
begin
FDlg.SetInitDate(AValue);
end;
end.
```

代码比较简单就不多解释了。

思考题：

1、做一个模仿 TcolorDialog 的对话框控件。

**Delphi 控件开发浅入深出（八）**

**八、数据敏感控件的制作。**

Delphi 的一大亮点就是它的数据库开发能力。而数据敏感组件则在这中间起着很重要的作用。在 Delphi 的 Data Control 页面下的控件都是用于显示和编辑数据库中的数据的。相信大家已经体会到数据敏感控件的好处了。我们这一节就给大家演示一下数据敏感控件的开发方法。

需要提醒大家的是，不像其他体系的控件，数据敏感控件并没有一个统一的基类，只要是从 TwinControl 类或其子类派生就可以，数据敏感控件的特殊之处就在于我们下面提到的数据连接。

相信用 Delphi 开发过数据库的人一定对 delphi 中没有一个日期数据敏感控件而恼火。每次都要我们自己处理数据的更新与显示。所以我们就来开发一个 DBDateTimePicker 控件。

新建一个控件，从 TdateTimePicker 派生，源代码如下：

```
{*******************************************************}
{ Linco TDBDateTimePicker
{ mail me: about521@163.com }
{*******************************************************}
unit DBDateTimePicker;
interface
uses
SysUtils, Classes, Controls, ComCtrls, DBCtrls, Messages, DB;
type
TDBDateTimePicker = class(TDateTimePicker)
private
FDataLink: TFieldDataLink;
procedure CMGetDataLink(var Msg: TMessage);message CM_GETDATALINK;
procedure DataChange(Sender: TObject);
procedure EditingChange(Sender: TObject);
procedure FSetDataField(AValue: string);
procedure FSetDataSource(AValue: TDataSource);
```

```
procedure FSetReadOnly(AValue: Boolean);
procedure ShowData;
procedure UpdateData(Sender: TObject);
function FGetDataField: string;
function FGetDataSource: TDataSource;
function FGetField: TField;
function FGetReadOnly: Boolean;
protected
procedure Change;override;
procedure Notification(AComponent: TComponent;Operation:
TOperation);override;
public
constructor Create(AOwner: TComponent); override;
destructor Destroy; override;
property Field: TField read FGetField;
published
property DataField: string read FGetDataField write FSetDataField;
property DataSource: TDataSource read FGetDataSource write FSetDataSource;
property ReadOnly: Boolean read FGetReadOnly write FSetReadOnly;
end;
procedure Register;
implementation
uses Variants;
constructor TDBDateTimePicker.Create(AOwner: TComponent);
begin
inherited Create(AOwner);
FDataLink := TFieldDataLink.Create;
FDataLink.OnDataChange := DataChange;
```

```
FDataLink.Control := self;
FDataLink.OnEditingChange := EditingChange;
FDataLink.OnUpdateData := UpdateData;
self.DateTime := Now();
end;
destructor TDBDateTimePicker.Destroy;
begin
FDataLink.Free;
inherited;
end;
procedure TDBDateTimePicker.CMGetDataLink(var Msg: TMessage);
begin
Msg.Result := Integer(FDataLink);
procedure TDBDateTimePicker.DataChange(Sender: TObject);
begin
if Field<>nil then
if Field.Value = null then
if (DataSource.DataSet.State = dsEdit)
or (DataSource.DataSet.State = dsInsert) then
Field.AsDateTime := Now();
ShowData;
end;
procedure TDBDateTimePicker.EditingChange(Sender: TObject);
begin
if (DataSource <> nil) and (DataField <> '') then
FDataLink.Edit;
end;
procedure TDBDateTimePicker.FSetDataField(AValue: string);
```

```
begin
FDataLink.FieldName := AValue;
end;
procedure TDBDateTimePicker.FSetReadOnly(AValue: Boolean);
begin
FDataLink.ReadOnly := AValue;
end;
procedure TDBDateTimePicker.ShowData;
begin
if (DataSource <> nil) and (DataField <> '') and(Field<>nil)then begin
case Kind of
dtkDate: if Field.AsString <> '' then
self.Date := Field.AsDateTime
else
self.Date := Now();
dtkTime: if Field.AsString <> '' then
self.Time := Field.AsDateTime
else
self.Time := Now();
else
self.DateTime := Now();
end;
end;
procedure TDBDateTimePicker.FSetDataSource(AValue: TDataSource);
begin
FDataLink.DataSource := AValue;
if AValue <> nil then
AValue.FreeNotification(self);
```

```
end;
procedure TDBDateTimePicker.Change;
begin
if (DataSource <> nil) and (DataField <> '') then
begin
FDataLink.Edit;
Field.Value := self.Text;
end;
inherited Change;
end;
procedure TDBDateTimePicker.Notification(AComponent: TComponent;Operation:
TOperation);
begin
if (Operation = opRemove) and (FDataLink <> nil) and
(AComponent = DataSource) then
DataSource := nil;
end;
procedure TDBDateTimePicker.UpdateData(Sender: TObject);
var
t: TFieldType;
begin
if (DataSource <> nil) and (DataField <> '') then
begin
t := FDataLink.Field.DataType;
case t of
ftTime: FDataLink.Field.AsDateTime := self.Time;
ftDate: FDataLink.Field.AsDateTime := self.Date;
ftDateTime: FDataLink.Field.AsDateTime := self.DateTime;
```

```
end;
end;
end;
function TDBDateTimePicker.FGetDataField: string;
begin
result := FDataLink.FieldName;
end;
function TDBDateTimePicker.FGetDataSource: TDataSource;
begin
result := FDataLink.DataSource;
end;
function TDBDateTimePicker.FGetField: TField;
begin
result := FDataLink.Field;
end;
function TDBDateTimePicker.FGetReadOnly: Boolean;
begin
result := FDataLink.ReadOnly;
end;
procedure Register;
begin
RegisterComponents('Linco', [TDBDateTimePicker]);
end;
end.
```

谈到开发数据敏感控件就不得不说数据连接(DataLink)，数据连接有很多种，开发数据敏感控件最常用到的就是字段数据连接（TFieldDataLink）。数据连接是联系数据敏感控件和数据库的通道。在数据敏感控件中就是凭借着数据连接来处理数据的更新和显示的。从后边我们的描述中您将更加能体会到，正是数据连接把数据在数据库中的表示反映到用户界面中，也是数据连接把数据从用户界面更新到数据库

中。数据连接就是一个“大媒人”（这其实是设计模式中 Mediator 中介者模式的典型应用）。

既然字段数据连接这么重要，我们就先来系统的介绍一下它吧！TfieldDataLink 闪亮登场！！！

TfieldDataLink 的属性：

（1）、property CanModify: Boolean;表示这个字段是不是只读的。

（2）、property Control: TComponent;指定这个字段数据连接被连接到哪个数据敏感控件。因为字段数据连接要把它的状态改变通知包含它的数据敏感控件。

（3）、property Editing: Boolean; 表示这个字段是不是可以被编辑。

（4）、property Field: TField;表示这个字段数据连接连接的字段。

（5）、property Active: Boolean;表示字段数据连接连接的数据集是否处于激活状态。

（6）、property FieldName: String;字段名。

（7）、property DataSource: TDataSource;表示它连接的数据源。

（8）、property DataSet: TDataSet;表示它负责维护的数据集。

方法：

（1）、function Edit: Boolean;尝试设置字段为编辑状态。如果设置成功则返回 True，反之返回 False;

事件：

（1）、property OnActiveChange: TNotifyEvent;当 Active 属性变化的时候发生此事件。

（2）、property OnDataChange: TNotifyEvent;当数据集发生变化的时候发生。

（3）、property OnEditingChange: TNotifyEvent;当数据源从编辑状态变为其他状态或从其他状态变为编辑状态的时候发生。

（4）、property OnUpdateData: TNotifyEvent;当向数据库提交对数据库的修改时发生此事件。

代码分析：

（1）、做为一个数据敏感控件，它首先要实现的功能就是允许用户将此控件连接到一个数据源(DataSource)。我们还要用户能选择这个控件绑定到哪个字段。

将控件连接到一个数据源，而数据源又是一个控件，所以这就是一个关联控件属性方法的应用。FsetDataSource 中 FDataLink.DataSource := AValue;这句代码是最重要的。就像我们前面讲到的数据连接就是一个在数据源和数据敏感控件之间的媒人，所以数据源(DataSource)要告诉媒人是它要被连接到数据敏感控件，而不是别人，告诉媒人的唯一方法就是设定媒人的 DataSource 为自己（即要绑定的数据源）。因为我们的显示日期的控件只能显示一个字段，还要告诉媒人自己的哪个字段要绑定到数据敏感控件，这个通过数据敏感控件的 FieldName 属性来进行。即：

procedure TDBDateTimePicker.FSetDataField(AValue: string);

begin

FDataLink.FieldName := AValue;

end;

（2）、我们还可以为控件增加一个 Field 属性，这样用户就可以通过 DBDateTimePicker.Field.AsString = ‘ok’;这样的方式对字段进行操作了。当然了，这最终还是通过数据连接的 Field 属性来进行的。

（3）、由于 VCL 内部通信机制的要求，数据敏感控件要响应 CM_GETDATALINK 事件。只要在事件相应函数里边把消息的 Result 域赋值为 DataLink 的地址就可以了。也就是：procedure TDBDateTimePicker.CMGetDataLink(var Msg: TMessage);

begin

Msg.Result := Integer(FDataLink);

end;

（4）、就像 DBEdit 一样，在用户通过改变控件中的日期时，应该能将改动保存到数据库字段中。我们覆盖控件的调度方法 Change（在显示的数据变化时被调用）以将变化保存到数据库中。

procedure TDBDateTimePicker.Change;

begin

if (DataSource <> nil) and (DataField <> '') then

begin

FDataLink.Edit;//设置数据连接为编辑状态，由这个媒人将数据库绑定的字//段设置为编辑状态

Field.Value := self.Text;//设定数据字段的值

end;

inherited Change;

end;

（5）、回头再来看看构造函数吧！

FDataLink.OnDataChange := DataChange;

FDataLink.OnEditingChange := EditingChange;

FDataLink.OnUpdateData := UpdateData;

FDataLink.Control := self;

前三句是设定响应数据连接事件处理句柄，正是这三句把数据库中的数据与用户界面联系了起来。关于这三个事件处理句柄的实现请参加源代码，这里就不多说了。

思考题：

1、做一个显示是/否的数据敏感控件，当这个控件与一个布尔类型的字段连接的时候，如果字段的值是 0 则显示“否”，如果字段的值是 1 则显示“是”；同时可以接受用户的修改，当用户在控件上单击一次鼠标，布尔值就翻转一次。

Delphi 控件开发浅入深出（结束语）

八讲的内容这么快就讲完了，通过这八讲相信大家对于 Delphi 控件开发有了大体的了解。但是仅仅了解还不够，要自己去发现：我做的程序中有没有可以提炼成控件的东西？一旦发现有可以提炼成控件的东西，就要尽力将它抽象化成控件。这样将能成倍甚至成十倍的提高系统的开发进度。

这篇文章其实是我去年写的，当时发在了我的个人网站上，但是由于空间到期了，我就一直没有理他。昨天注册了一个 blog，所以想把这些东西再放出来，希望能给需要它的朋友带来一点帮助。

我在实际的开发系统中开发了很多实用的控件，但是由于 blog 空间的限制没有办法把他们上传上来，需要的可以给我留言或者给我发送 email。

如何用 Delphi 编写自己的可视化控件

可视化控件(Visual Component)实际上就是一个类(class)，要编写一个类，可以直接在*.pas 文件中编写。但是要编写控件，则必须使用包(package)。从 File 菜单中选择 New，新建一个 Package，这就是存放和安装控件用的包。然后单击 Package 窗口中的 Add 按钮，添加一个元件(Unit)。

在弹出的对话框最上方选择 New Component。因为一个控件的所有属性、方法、事件不可能都由自己编，所以就需要选择祖先类（或者叫做"父类"或"基类"），然后再在其上面添加自己的属性、方法、事件。在 Ancestor type 后的下拉框中选择所需的祖先类。由于编写可视化控件必须要画图，所以选择 TGraphicControl 作为祖先类。再在 Class Name 框中输入新控件（类）的名称，一般以"T"开头。Palette Page 是用来选择新控件在 Delphi 的窗口中的控件页面名称，例如"Standard"，这

个可以自己取。在 Unit File Name 中添好新控件文件的路径及文件名，单击 OK 按钮。新的控件便加入了。现在可以为该控件编写代码了。

下面以编写一个可以自定义图片的滚动条为例，说明编写可视化控件的方法。按照上面的方法，选择 TGraphicControl 为祖先类，新控件的名称是 TPigHorizontalScroller （小猪水平滚动条）。选择好文件路径和文件名后，单击 OK 按钮，开始编写代码。

每一个控件，都会被创建（Create）和删除（Destroy），所以必须首先编写这两个过程。对于控件中的每一个过程，都必须在前面先定义，然后再在后面编写。定义的过程或属性有三种：一、在 private 后定义的是属于控件内部使用的，使用该控件的人无法看到；二、在 protected 后定义的一般是看不到的，只在别人使用该控件作为祖先类编写其它控件时才可见；三、在 public 后定义的只允许别人在程序中调用；四、在 published 后定义的可以在属性窗口（Object Inspector）中看到。由于创建和删除过程除了在编程过程中建立控件时自动执行外，还可能在程序运行过程中动态创建控件时被调用，所以把它定义在 public 后⑴。（该序号表示次步骤在所附源程序中的代码的位置，下同）现在也许还不知到应该在这两个过程中编写什么，如何去编。我们在下面将会讲到。

我们首先为这个控件添加一些属性。我们定义一个 Max 属性用于设置或读取滚动条的最大值。因为在程序中一般不直接使用属性，所以要定义一个变量，和该属性对应起来，一边修改或读取其值。因为它只在控件内部使用，所以我们把它定义在 private 后⑵。（一般与属性相关联的变量都以"F"开头，例如 FMax）定义好变量后，再定义属性。这个属性需要再 Object Inspector 窗口中可见，所以把它定义再 published 后⑶。定义的语法是：

property <属性名>:<类型> read <读取该属性时对应的变量> write <写入该属性时对应的变量或过程>

其它的变量和属性也类似的定义（例如 Min 最小值、Value 当前值等）。下面我们定义几个属性和变量，用于设置滚动条的图片（因为图片变量比较特殊，所以单独讲一下）。我们把 LeftButtonUpPicture（向左按钮图片）、LeftButtonDownPicture（向左按钮按下图片）

等定义为 TBitmap 类型（一定要定义相对应的变量）。

大家一定注意到了，在所附的源程序中，定义这几个属性时，read 后所指定的读取属性时对应的变量是 F...，而 write 后指定的写入该属性时对应的不是一个变量，而是一个 Set...之类的东西，这是一个自定义的过程。作为该功能的过程的定义为：

procedure <过程名>(Value: <被设置的属性的值的类型>)

因为执行写入该类属性的时候需要做其它的事情，所以不能光用一个变量来处理，应该用一个过程来处理。这中过程一般定义在 protected 后。在该类过程中，使用

一个在⑷处这样一个语句来给 TBitmap 类型的变量来赋值，这是由于该类型的变量不能直接赋值而采用的。

定义完这些 TBitmap 类型的变量的属性后，上面讲的 create 过程和 destroy 过程中就需要编写代码了。因为 TBitmap 也是一个类，所以在 create 过程中必须创建⑸，在 destroy 过程中必须释放掉(free)⑹。这里⑺所指的 inherited 语句是用于指明该过程是从祖先类类中继承来的。（这个一定不能掉）。

因为我们编写的是可视化控件，所以必须在控件上画图。我们这个控件的祖先类 TGraphicControl 中封装有一个 Canvas（画布）对象，我们可以直接使用它来画图。如果你对画布的使用还不太熟悉，最好去找一本书来看一看。

下面要做的工作就是画图了。如何在控件上画图呢？祖先类 TGraphicControl 中有一个 Paint 事件，当控件需要重画时便会自动触发。我们现在要做的就是要为这个事件编写一段程序。首先在 protected 后定义一个 Canvas 对象。由于它是祖先类中已有的，所以不需要加任何说明⑻。我们将使用这个对象来画图。接着，就要定义一个 Paint 过程，编写绘制控件的代码。先在 public 后定义 Paint 过程。由于它是由祖先类触发的，而不是由用户调用的，所以后面必须加上 override，否则，该控件将会由于 Paint 过程永远不会被调用而不成为可视化控件⑼。下面我们就来编写 Paint 过程的代码⑽。

该文章所附的源程序的 Paint 过程中的 T_Height 等变量是用来保存滚动条中按钮、滑块等的大小的，这部分程序和普通的 Application 中的程序差别不大，大部分都是对画布进行操作，相信大家一看就明白。值得注意的是下面对 FAutoSize 变量的判断⑾,FAutoSize 是和该控件的属性 AutoSize 相关联的布尔型变量，是用来设置这个控件的大小是否随图片的大小而变化的。注意，在控件的代码中，一般都不直接调用属性，而是使用与其相对应的的变量。

程序编到这里，就算是终于给自己的新控件做了一个外型了，不过它还不能滚动。现在我们来编写鼠标事件，让我们能够操纵它。鼠标事件的过程的定义和 Paint 过程很相似，只是后面要加上参数说明⑿,鼠标事件分为 MouseDown、MouseMove 和 MouseUp 三个，在定义后面都要加上 override。接下来在后面编写它的代码。注意：这里的鼠标事件是 Mouse...，而不是通常的 OnMouse...。可是在⒀处的定义是干什么用的呢？这里的事件定义，都是给用户使用的，也就是说，当使用该控件时，会在 Object Inspector 中的 Event 页面中显示出来。

这些鼠标事件的代码也非常简单，判断鼠标的坐标，在画布上画出相应的图片等，并同时触发相应的事件。值得注意的是，在调用自定义事件时，都要先用⒁处的这样一个语句来判断用户是否已经为该事件编写代码。这一点非常重要，否则会调用出错。

大家注意到了，刚才所调用的事件都是自定义的，定义的方法也很简单，和定义属性差不多，只是类型时 TNotifyEvent 罢了。

TNotifyEvent 是默认事件，其定义为：

TNotifyEvent = procedure(Sender: TObject)

如果你要定义另外形式的事件，就必须这样：先在 type 后编写

<事件类型名称> = procedure(<参?gt;:<类型>)

例如：

TCustomEvent = procedure(a: Integer; b:String);

然后在 public 后定义：

<事件名称>:<事件类型名称>

例如：

AnEvent: TCustomEvent;

看完这些，这整个程序你应该理解了吧。如果编译或运行出错，注意检查以下几点：

１、create 和 destroy 过程中是否有 inherited 语句；

２、TBitmap 类型的变量 create 和 free 了没有；

３、过程前有没有控件名，例如：TPigHorizontalScroller.MoseMove

判断鼠标是否进入或离开控件的方法：

定义如下的过程：

procedure MouseEnter(var Msg: TMessage); message CM_MOUSEENTER;

procedure MouseLeave(var Msg: TMessage); message CM_MOUSELEAVE;

再在下面编写代码就行了。这个方法用于编写三态按钮很有用。