Curso de criação de componentes em Delphi

Unidade 9. Editores de Propriedades (II).

Voltar ao índice

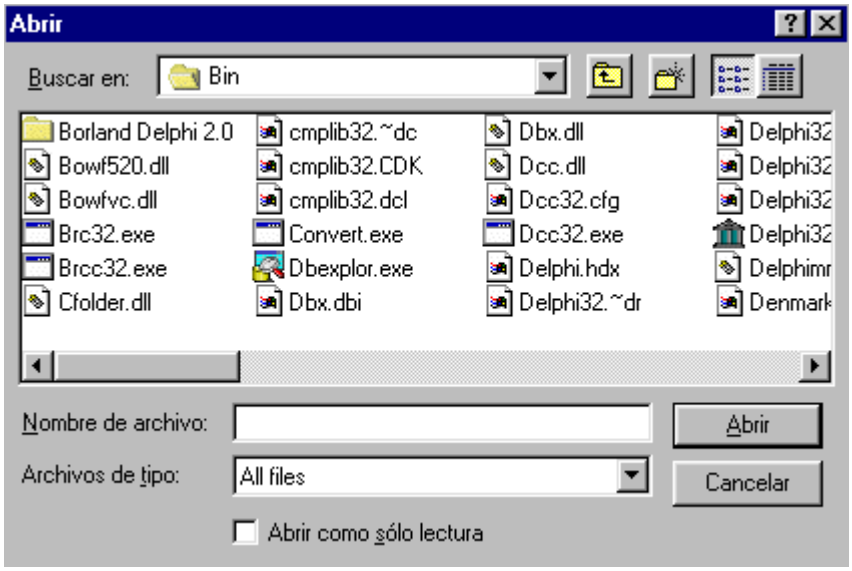
Por Luis Roche

Introdução

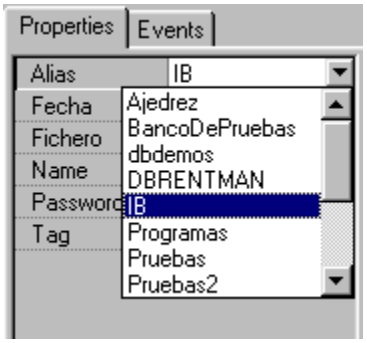
Na unidade anterior nós aprendemos a operação básica de editor de propriedades e nós desenvolvemos um exemplo de editor de propriedades que trabalhava no Object Inspector (BinaryPropEd).

Nesta unidade nós desenvolveremos quatro editores de propriedades e um componente que será responsável por eles.

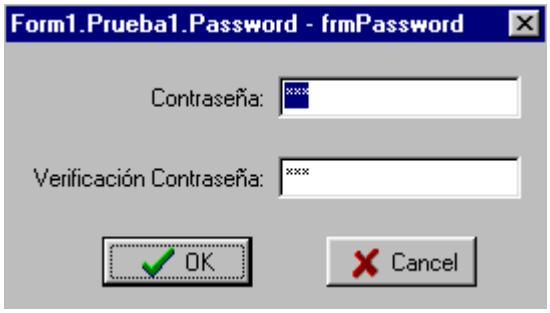
> **TArqProperty é** um editor de propriedades do tipo diálogo. Em alguma ocasião nós desenvolveremos um componente em qual das propriedades deles deveria armazenar o nome de um arquivo. Nós obrigaremos o usuário sofrer fazendo uma rotina para escrever um nome de arquivo correspondente? Claro que não! TArqProperty mostrará para o usuário o diálogo típico OpenDialog de forma que por meio do editor de propriedades que simplificará o processo de pegar o nome de um arquivo: Não esqueça que a principal tarefa de um editor de propriedades é facilitar a vida do programador.



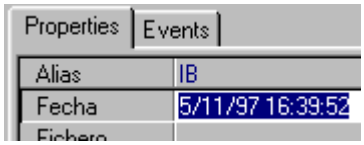
- **TAliasProperty é** editor de propriedades inteligente de valores da propriedade identicos ao valor da propriedade DatabaseName do componente TTable: permite a escolha de um alias de banco de dados por meio de uma lista drop-down no próprio Object Inspector.



- Aprofundando um pouco mais no tópico dos editores de propriedades tipo caixas de diálogo nós criaremos um partindo do zero, quer dizer, sem usar uma caixa de diálogo predefinida como no caso do TArqProperty publicado. Concretamente, nós criaremos um editor que nos permite a introdução de contra-senhas. Imagine isso em uma propriedade de um componente que nós precisamos manter um valor de string do tipo (password). Se nós não criássemos qualquer editor de propriedades, o usuário do componente escreverá o valor diretamente no Object Inspector , mas o que se escreve será completamente visível (não aparecerão como acontece com o componente TEdit). Não muito elegante, verdade? Assim nós criaremos uma caixa de diálogo que nos permitirá escrever e verificar a contra-senha para validar à propriedade. Este será nosso **TPasswordProperty** .



- **Por ultimo criaremos um editor de propriedades, TDateTimeProperty,para a introdução de datas. Deste modo nós poderemos introduzir datas em uma propriedade de tipo TDateTime (nós nos lembramos que as datas são do tipo Ponto Flutuante) em vez de ter que introduzir o número equivalente.**



Parece interessante, verdade? Mãos para o trabalho:)

Um componente de teste

Antes de começar a desenvolver os editores de propriedades nós criaremos um componente que nos permitirá os provar de acordo como quisermos. Este componente se chamara TPrueba e seu código é:

```
unit Unidad9;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  DsgnIntf, DB, PasswordForm;

type
  TPrueba = class(TComponent)
  private
    FArquivo : string;
    FAlias : string;
    FData : TDateTime;
    FPassword : string;
  protected
  public
    constructor Create(AOwner : TComponent); override;
  published
    property Arquivo : string read FFichero write FFichero;
    property Alias : string read FAlias write FAlias;
    property Data : TDateTime read FFecha write FFecha;
    property Password : string read FPassword write FPassword;
  end;

...
```

```
implementation

...
constructor TPrueba.Create(AOwner : TComponent);
begin
  inherited Create(AOwner);
  FDAta:=Now;
end;

...
```

Qualquer coisa do outro mundo. Quatro propriedades de cada um dos editores de propriedades para testarr e um contrutor comum para colocar o valor padrão de data.

Serve como aviso que pela simplicidade do componente nós desenvolveremos o componente e os quatro editores de propriedades em uma única unidade (há uma unidade adicional necessária para o TPasswordProperty). Como eu digo, nós faremos isto desta forma pela simplicidade, mas não é a forma mais correta. Em geral, cada editor de propriedades deveria entrar em uma unidade independente do próprio componente, como nós já vimos na unidade 8.

O editor de propriedades TArqProperty

Eu fixei na propriedade tal Arquivo e como nós implementamos isto no componente TPrueba. Claro que, é do tipo string. Nós também não definimos nenhum métodos Set Get, pois o read/write de valores nesta propriedade é feito diretamente no campo associado (FArquivo). Tal como é, quando um usuário do componente quer introduzir um valor nesta propriedade, ele terá que escrever isto à mão, o que pode acabar sendo uma tarefa tediosa se o arquivo em questão esta em um caminho longo. De forma que, por generosidade nossa, nós decidimos o dar uma mão: nós construiremos um editor de propriedades para ele.

O Delphi já incorporá um componente que administra isso muito bem para a de arquivos: TOpenDialog, de forma que nós veremos como usar isto melhor em nosso editor.

O primeiro passo é decidir de quem nosso editor se derivará. Em nosso caso, desde que a propriedade arquivo é do tipo string, nós decidimos herdar de TStringProperty. Logo nós deveremos decidir se a propriedade será editavel no próprio editor de objetos ou usará uma caixa de diálogo. Neste caso, sem dúvida nós deveremos escolher a segunda opção. Claro que isto não tira a chance do escrever o valor da propriedade diretamente no Object Inspector.

Agora então, como indicar ao Delphi que se trata de um editor de propriedades do tipo caixa de dialogos? Para isto, nós devemos usar outro método da classe base de todos os editores de propriedades (TPropertyEditor): o método **GetAttributes** Este método é uma função que determina que características terá o editor de propriedades. As características queridas deveriam os devolver como resultado desta função; este resultado é do tipo **TPropertyAttributes** O tipo TPropertyAttributes é um grupo (Set) isso pode levar os valores seguintes:

Atributo	**Descrição**
paValueList	Especifica que o editor devera mostrar uma lista com os possíveis valores pela propriedade. Para encher a lista o método GetValues é usado
paSortList	Só válido se paValueList é selecionado. Especifica que a lista de valores será

	mostrada ordenada.
paSubProperties	Indica que o editor define subpropiedades para mostrar à direita (p.e. a propridade fonte do TForm usa isto). Para gerar a lista de propriedades o método GetProperties é usado.
paDialog	Indica que o editor de propriedades devera mostrar uma caixa de diálogo em resposta para ao método Edit. (p.e. a propriedade glyph de um TSpeddButton). Deste modo quando selecionando a propriedade aparecerá um botão com '... '.
paMultiSelect	Possibilita a multi-seleção de opções
paAutoUpdate	O Object Inspecto o método SetValue toda vez que o valor da propriedade é mudado.
paReadOnly	O usuário não pode modificar o valor da propriedade
paRevertable	Especifica se a propriedade pode recuperar seu valor original

Devido a este quadro, é fácil de implementar o método GetAtributes em nosso editor de propriedades:

```
 interface
 ...
 TArqProperty = class(TStringProperty)
 public
   function GetAttributes : TPropertyAttributes; override;
   procedure Edit; override;
 end;
 ...

function TArqProperty.GetAttributes : TPropertyAttributes;
begin
 Result:=[paDialog];
end;
```

Novamente por causa da simplicidade nós só ativamos o atributo paDialog, é claro que nós poderíamos ativar outro atributo como paMultiSelect, etc. Neste caso, escreveriamos [paDialog, paMultiSelect]

Nós só saberemos que o Delphi ativará a caixa de diálogo, quando o usuário clica no botão '... ' ou da um duplo clique na propriedade,o Delphi invocará o método Edit de nosso editor de propriedades. Neste método nós deveremos colocar o codigo necessario para mostrar a caixa de dialogo de abertura de arquivos e colocar o nome do arquivo na propriedade se o usuário clicar no OK.. Este método que nós devemos reimplementar (overridee, ver a seção de interface), é deste modo:

```
...

procedure TArqProperty.Edit;
var
  OpenDialog : TOpenDialog;   {TOpenDialog está na unidade Dialogs, que esta na clau
begin
  OpenDialog:=TOpenDialog.Create(Application);   {Criamos a caixa de diálogo}
  try
    OpenDialog.Filter:='All files|*.*';    {Colocamos suas propriedades iniciais}
    if OpenDialog.Execute then   {Se o usuário clica OK...}
      SetStrValue(OpenDialog.FileName);   {Colocamos o novo valor na propriedade}
  finally
    OpenDialog.Free;   {Liberamos a caixa de diálogo}
  end;
```

```
end;

...
```

Mais fácil impossível. Só temos que comentar que para termos certeza da liberação de recursos (neste caso da caixa de diálogo) usamos o bloco try... finally **.**

É só! Com 10 ou 15 linhas de código nós aliviamos o pobre usuário de ficar sofrendo com isso tudo. Nós agora registraremos o editor:

```
procedure Register;
begin
   ...
  RegisterPropertyEditor(TypeInfo(string),TPrueba,'Arquivo',TArqProperty);
  ...
end;
```

O editor de propriedades TAliasProperty

Vamos contruir agora um editor de propriedades para a propriedade Alias. Como nós já sabemos, o componente TTable tem uma propriedade denominada DatabaseName que especifica o alias que conecta ao banco de dados. Selecionar um valor por esta propriedade é muito facil, basta escolhermos um na lista. Este é o comportamento que nós queremos para a nossa propriedade Alias. Nós poderíamos procurar no codigo fonte do VCL e tentar também registrar o editor de propriedades da propriedade DatabaseName de forma que isto incluiria nossa nova propriedade, mas como a construção de um editor deste tipo é muito simples, nós mesmos desenvolveremos isto.

A seção de interface de nosso editor é o seguinte:

```
...
TAliasProperty = class (TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    procedure GetValues(Proc : TGetStrProc); override;
  end;
...
```

O método GetAttributes conhecido na seção anterior. Basta colocar o atributo paValueList (indica a lista de opções) e o paSortList (indica que as opções estarão ordenadas alfabeticamente).

```
function TAliasProperty.GetAttributes : TPropertyAttributes;
begin
    Result:=[paValueList, paSortList];
end;
```

Agora nós preencheremos a lista de aliases. Para isto, reimplementaremos (override) no método **GetValues. Este método recebe um único parâmetro: um ponteiro para método. Realmente este ponteiro realmente da referência ao metodo interno Add, que adiciona uma string na lista. Os diversos elementos inseridos na lista pelo método GetValues que invoca o método refereciado pelo ponteiro e transforma isto em um valor do tipo string. Soa complicado, não? Não se preocupe com isso, pois não é tão dificil; so significa que temos que usar a senteça Proc(string) para cada elemento a adicionar à lista. Em nosso caso, como nós queremos adicionar os nomes dos aliases existentes, nós faremos uma ligação a Proc(nome do alias) e adiciona todos os valores.**

Previamente, nós teremos obtido esses aliases existente por meio do método GetAliasList do objeto TSession:

```
procedure TAliasProperty.GetValues(Proc : TGetStrProc);
Var
  AliasList : TStringList;   {lista com os alias existentes}
  i : integer;
begin
  try
    AliasList := TStringList.Create;   {Criamos a lista}
    Session.GetAliasNames(AliasList);   {Obtemos os alias existentes}
    for i:=0 to AliasList.Count - 1 do     {Para cada alias...}
      Proc(AliasList[i]);   {...fazemos a chamada ao método Proc}
  finally
    AliasList.Free;   {Liberamos a lista}
  end;
end;
```

Com isto nós já construímos já nosso novo editor de propriedades. Só falta registrá-lo:

```
procedure Register;
begin
   ...
  RegisterPropertyEditor(TypeInfo(String),TPrueba,'Alias',TAliasProperty);
  ...
end;
```

O editor de propriedades TPasswordProperty

Vamos construir um outro editor de propriedades de caixa de dialogos. A operação dela será semelhante ao TArqProperty. Nós colocaremos paDialog para indicar que é editor de caixa de diálogo e no método Edit estara todo o codigo do componente.

A diferença principal é que nosso componente não usará uma caixa de dialogo existente, nós criaremos tambem uma caixa de dialogos, com dois labels e dois edit boxes:

A coisa mais importante é nomear à propriedade PasswordChar do dois TEdit, para o caráctere ' * ' de forma que este caráter aparece quando um usuário digita uma contra-senha. Também, no evento OnCloseQuery conferirá a validade da contra-senha introduzida.

Os Edits foram nomeados de PW1 e PW2

O código da unidade PasswordForm:

```
unit PasswordForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons;

type
  TfrmPassword = class(TForm)
    lpwd: TLabel;
    lVpwd: TLabel;
    PW1: TEdit;
    PW2: TEdit;
    bOK: TBitBtn;
    bCancel: TBitBtn;
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
  private
  public
  end;

var
  frmPassword: TfrmPassword;

implementation

{$R *.DFM}

procedure TfrmPassword.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
begin
  if (ModalResult=mrOK) then
    if (PW1.Text = '') then
    begin
      ShowMessage('Deve se inserir a contra-senha');
      CanClose:=False;
     end
  else if (ModalResult=mrOK) and (PW1.Text <> PW2.Text) then
   begin
    ShowMessage('Verificação inválida. Por favor retente');
    CanClose:=False;
   end;
end;

end.
```

Ya hemos construido el cuadro de diálogo, ahora sólo nos resta "engancharlo" al edit
propiedades. Para ello, efectuamos la llamada al form en el método Edit del editor d
Veamos como queda el código del editor:

```
function TPasswordProperty.GetAttributes : TPropertyAttributes;
begin
  Result:=[paDialog];
end;

function TPasswordProperty.GetValue : string;
begin
```

```
  Result:=Format('(%s)',[GetPropType^.Name]);
end;

procedure TPasswordProperty.Edit;
begin
  frmPassword := TfrmPassword.Create(Application);
  try
    frmPassword.Caption:=GetComponent(0).Owner.Name+'.'+
      GetComponent(0).Name+'.'+GetName+' - '+
      frmPassword.Caption;
    frmPassword.PW1.Text:=GetStrValue;
    frmPassword.PW2.Text:=frmPassword.PW1.Text;
    if frmPassword.ShowModal = mrOK then
      SetStrValue(frmPassword.PW1.Text)
  finally
    frmPassword.Free;
  end;
end;
```

Nós já construímos a caixa de diálogo, agora só nos falta encaixa-lo no editor de propriedades. Para isto, nós fazemos a ligação à forma com o método Edit do editor de propriedades. Vamos ver como é o codigo:

```
function TPasswordProperty.GetAttributes : TPropertyAttributes;
begin
  Result:=[paDialog];
end;

function TPasswordProperty.GetValue : string;
begin
  Result:=Format('(%s)',[GetPropType^.Name]);
end;

procedure TPasswordProperty.Edit;
begin
  frmPassword := TfrmPassword.Create(Application);
  try
    frmPassword.Caption:=GetComponent(0).Owner.Name+'.'+
      GetComponent(0).Name+'.'+GetName+' - '+
      frmPassword.Caption;
    frmPassword.PW1.Text:=GetStrValue;
    frmPassword.PW2.Text:=frmPassword.PW1.Text;
    if frmPassword.ShowModal = mrOK then
      SetStrValue(frmPassword.PW1.Text)
  finally
    frmPassword.Free;
  end;
end;

Só há uma coisa nova: no método GetValue nós não queremos que seja mostrado

o valor da contra-senha, então nós poderíamos mostrar outra coisa. Poderia

ser uma cadeia de asteriscos...

procedure Register;
begin
  ...
```

```
  RegisterPropertyEditor(TypeInfo(String),TPrueba,'Password',TPasswordProperty);
end;
```

O editor de propriedades TDateTimeProperty

Para terminar esta unidade nós desenvolveremos um editor de tipo de propriedades TDateTime.

Como nós já sabemos o tipo TDateTime do Delphi é do tipo ponto flutuante, de forma que se nós temos em um componente com uma propriedade de tipo DateTime, e nós não registramos qualquer editor de propriedades para esta propriedade, o usuário terá que escrever a data querida em formato decimal. Que ruim! Principalmente se nós pensamos que com seis linhas de código é resolvido o problema:

```
function TDateTimeProperty.GetValue : string;
begin
  Result:=DateTimeToStr(GetFloatValue);
end;

procedure TDateTimeProperty.SetValue(const Value : string);
begin
  SetFloatValue(StrToDateTime(Value));
end;

procedure Register;
begin
  ...
  RegisterPropertyEditor(TypeInfo(TDateTime),TPrueba,'Fecha',TDateTimeProperty);
end;
```

Nada novo até aqui. Nós só nos limitamos para chamar os métodos GetFloatValue e SetFloatValue.

Conclusões

Com isto que nós terminamos o tópico dos editores de propriedades. Nós aprendemos a criar e publicar editores editaveis no Object Inspector, que pode ser preenchido diretamente na propriedade ou por uma caixa de dialogo. O poder de editor de propriedades é imenso: por meio do métodos SetValue e GetValue podemos fazer uma verificação do valor da propriedade e agir depois. Também, nosso editor pode fazer mil coisas: consultar um arquivo ini, fazer cálculos complexos ou até mesmo operar com um banco de dados!

Mas os editores de propriedades têm duas limitações:

- Eles só operam em design-time, aspecto que nós deveríamos nos lembrar que tem coisas que o usuário pode querer fazer em run-time. Um exemplo claro é as propriedades de tipo Lista p.e. os artigos de propriedade de um TListBox. Em design-tmie o editor de propriedades especifico permite o usuário de um modo intuitivo adicionar e eliminar elementos. Em troca, em run-time, ele tem os métodos Add e Delete para operar com a lista.
  Editor de propriedades **só** opera com o valor de **uma** propriedade. Quer dizer, nós não podemos alterar o valor de propriedades diferentes ao mesmo tempo, desde os métodos GetValue e SetValue fazem referência à propriedade que está publicada, e não para o resto deles. Então, quando nós queremos alterar várias propriedades de um componente ao mesmo tempo, nós usaremos **Editor de componentes**, tópico isto que será objeto de nossa próxima unidade.

Codigo fonte dos editores de propriedades.

```
 unit Unidad9;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  DsgnIntf, DB, PasswordForm;

type
  TPrueba = class(TComponent)
  private
    FArquivo : string;
    FAlias : string;
    FData : TDateTime;
    FPassword : string;
  protected
  public
    constructor Create(AOwner : TComponent); override;
  published
    property Arquivo : string read FFichero write FFichero;
    property Alias : string read FAlias write FAlias;
    property Data : TDateTime read FFecha write FFecha;
    property Password : string read FPassword write FPassword;
  end;

  TArqProperty = class(TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    procedure Edit; override;
  end;

  TAliasProperty = class (TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    procedure GetValues(Proc : TGetStrProc); override;
  end;

  TDateTimeProperty = class(TFloatProperty)
    function GetValue : string; override;
    procedure SetValue(const Value  : string); override;
  end;

  TPasswordProperty = class(TPropertyEditor)
    function GetAttributes : TPropertyAttributes; override;
    function GetValue : string; override;
    procedure Edit; override;
  end;

procedure Register;

implementation

constructor TPrueba.Create(AOwner : TComponent);
begin
  inherited Create(AOwner);
  FFecha:=Now;
end;
```

```
function TFicheroProperty.GetAttributes : TPropertyAttributes;
begin
  Result:=[paDialog];
end;

procedure TArqProperty.Edit;
var
  OpenDialog : TOpenDialog;
begin
  OpenDialog:=TOpenDialog.Create(Application);
  try
    OpenDialog.Filter:='All files|*.*';
    if OpenDialog.Execute then
      SetStrValue(OpenDialog.FileName);
  finally
    OpenDialog.Free;
  end;
end;


function TAliasProperty.GetAttributes : TPropertyAttributes;
begin
    Result:=[paValueList, paSortList];
end;

procedure TAliasProperty.GetValues(Proc : TGetStrProc);
Var
  AliasList : TStringList;
  i : integer;
begin
  try
    AliasList := TStringList.Create;
    Session.GetAliasNames(AliasList);
    for i:=0 to AliasList.Count - 1 do
      Proc(AliasList[i]);
  finally
    AliasList.Free;
  end;
end;

function TDateTimeProperty.GetValue : string;
begin
  Result:=DateTimeToStr(GetFloatValue);
end;

procedure TDateTimeProperty.SetValue(const Value : string);
begin
  SetFloatValue(StrToDateTime(Value));
end;

function TPasswordProperty.GetAttributes : TPropertyAttributes;
begin
  Result:=[paDialog];
end;

function TPasswordProperty.GetValue : string;
begin
  Result:=Format('(%s)',[GetPropType^.Name]);
end;
```

```
procedure TPasswordProperty.Edit;
begin
  frmPassword := TfrmPassword.Create(Application);
  try
    frmPassword.Caption:=GetComponent(0).Owner.Name+'.'+
      GetComponent(0).Name+'.'+GetName+' - '+
      frmPassword.Caption;
    frmPassword.PW1.Text:=GetStrValue;
    frmPassword.PW2.Text:=frmPassword.PW1.Text;
    if frmPassword.ShowModal = mrOK then
      SetStrValue(frmPassword.PW1.Text)
  finally
    frmPassword.Free;
  end;
end;

procedure Register;
begin
  RegisterComponents('Pruebas', [TPrueba]);
  RegisterPropertyEditor(TypeInfo(string),TPrueba,'Arquivo,TArqProperty);
  RegisterPropertyEditor(TypeInfo(String),TPrueba,'Alias',TAliasProperty);
  RegisterPropertyEditor(TypeInfo(TDateTime),TPrueba,'Data',TDateTimeProperty);
  RegisterPropertyEditor(TypeInfo(String),TPrueba,'Password',TPasswordProperty);
end;

end.




unit PasswordForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons;

type
  TfrmPassword = class(TForm)
    lpwd: TLabel;
    lVpwd: TLabel;
    PW1: TEdit;
    PW2: TEdit;
    bOK: TBitBtn;
    bCancel: TBitBtn;
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
  private
  public
  end;

var
  frmPassword: TfrmPassword;

implementation

{$R *.DFM}

procedure TfrmPassword.FormCloseQuery(Sender: TObject;
```

```
  var CanClose: Boolean);
begin
  if (ModalResult=mrOK) then
    if (PW1.Text = '') then
    begin
      ShowMessage('Debe introducir una contraseña');
      CanClose:=False;
     end
  else if (ModalResult=mrOK) and (PW1.Text <> PW2.Text) then
   begin
    ShowMessage('Verificación fallida. Por favor reintente');
    CanClose:=False;
   end;
end;

end.
```

Luis Roche *revueltaroche@redestb.es*

Ultima modificación 06.09.1997