Curso de criação de componentes em Delphi

Unidade 8. Editores de Propriedades (I).

Voltar ao índice

Por Luis Roche

Nesta unidade nós aprenderemos a criar nossos próprios editores de propriedades. Os editores de propriedades são uma ferramenta poderosa que fazem diferença entre um componente simplesmente aceitável e um realmente bom. Por eles nós poderemos dotar a nossos componentes de características novas que vão além da validação do valor das propriedades. Graças a eles nós faremos a vida mais fácil para os programadores que usam nossos componentes, e isso sempre é bom não é?

Introdução aos editores de propriedades

Até agora nós se criamos os componentes sem se preocupar como foram introduzidos os valores das diferentes propriedades definidas neles. Chegou o momento de estudarmos este aspecto. Como já sabemos, para inserir e ler o valor das propriedades em design-time nós usamos o Object Inspector. O Object Inspector se divide em duas colunas: no da direita estão os nomes das propriedades e no da esquerda seus valores. Introduzimos um valor novo e ele aparece. Mas isto é só aparência. Os componentes se interagem com o Object inspector através dos editores de propriedades. Do ponto de vista do usuário, o Object Inspector é o responsável de publicar as propriedades, mas por trás há uma série de objetos, os editores de propriedades que se encarregam de definir as capacidades de edição das diversas propriedades de um componente.

Em design-time, quando você seleciona um componente, o Object Inspector cria instâncias dos editores de propriedades necessários para publicar as propriedades definidas no componente selecionado. Quando termina a edição, o mesmo Object Inspector destroi os editores de propriedades criados.

O Delphi tem uma série de editores de propriedades padrões que são o bastante para a maioria das propriedades com que nós habitualmente trabalhamos. Mas como sempre, o Delphi é tão potente que permite que nós criemos nossos próprios editores que podem substituir certas propriedades que estão por padrão. Pobre VB! ;)

Os deveres do Editor de Propriedades

O Editor de propriedades deve dar resposta a dois aspectos principais:

- Definir como a propriedade devera ser publicada. Aqui há duas possibilidades: pode ser modificado o valor da propriedade no próprio Object Inspector ou você pode usar uma caixa de diálogo para adicionar mais flexibilidade à edição (p.e. propriedade cor)
- Converter o valor da propriedade para um valor tipo string. O Object Inspector **sempre trabalha com Strings.** Embora a propriedade seja do tipo inteiro ou Ponto Flutuante, o Object Inspetor só usa o tipo String em suas propriedades. É o editor de propriedades que faz todo trabalho na representação desta propriedade. Traduzindo: pode ir da coisa mais simples (usar a função IntToStr) até a uma mais complexa (programar uma rotina para a conversões). Tudo dependerá do tipo de propriedade com que nós estamos trabalhando.

Passos necessários para escrever um editor de propriedades

Os passos necessários para escrever um editor de propriedades são o seguinte:

- Criar uma nova unidade na qual nós definiremos o editor de propriedades. Mais tarde nós falaremos mais amplamente sobre este ponto, desde que não é tão trivial quanto pode parecer em princípio;)
- Colocar a unidade **DsgnIntf** para a clausula uses do editor de propriedades. Nesta unidade estão definidos os editores de propriedades padrões que o Delphi usa, além da importante classe TPropertyEditor,que é a base de todos os editores de propriedades.
- Criar uma classe nova que descende de TPropertyEditor ou de algum do descendentes deles. Por convenção, o nome dos editores de propriedades terminam com a palavra Property. P.e. TIntegerProperty, TStringProperty... Para os editores principais de propriedades logo são mostrados por padrão no Delphi.

Editor de propriedades	**Tipo**
TPropertyEditor	Bases de classe para todos os editores de propriedades
TIntegerProperty	Byte, word, integer, Longint,
TCharProperty	Char
TEnumProperty	Tipos enumerados
TSetProperty	Sets
TFloatProperty	Single, Doble, Extended, Comp, Currency,
TStringProperty	Strings
TClassProperty	Qualquer objeto
TMethodProperty	Qualquer método (eventos)
TComponentProperty	Para propriedades que fazem referência a componentes

- Implementar os métodos necessários para adicionar ao editor de propriedades as funcionalidades queridas.
- Registrar o editor de propriedades na VCL

Mais tarde nós nos aprofundaremos em todos os aspectos dos outros editores de propriedades, vamos criar agora o nosso primeiro editor de propriedades.

Editor de propriedades para números binários

Nós desenvolveremos nosso primeiro editor de propriedades que

será editavel no próprio Object Inspector. Imagine que nós criamos um componente com uma propriedade do tipo inteiro que tenha um valor para ser visualizado em base binária e não em base decimal.. Se nós não criássemos editor de propriedades e deixássemos que o editor de propriedades padrão do Delphi (TIntegerProperty no caso) fizesse o trabalho a visualização seria em base decimal e não em base binária como nós queremos.. Isto é típico caso no qual desenvolvendo um editor de propriedades simples se economiza muito trabalho para o futuro usuário do nosso componente e evita que ele tenha que fazer a conversão entre decimal e binário.

Como nós já comentamos, nós temos que decidir que de classe se derivará o nosso editor. Em nosso caso é fácil, a propriedade armazenará um valor do tipo inteiro, assim nós usaremos IntegerProperty

Muito bem, nossa propriedade armazena um inteiro, mas nós não queremos que o Object Inspector nos mostre diretamente o valor decimal dele, nós queremos fazer uma conversão deste valor de decimal a base binária e que só então o Object Inspetor nos mostrará o valor. Para fazer isto nós implementaremos (override) na função **GetValue. Esta função, definida em TPropertyEditor, é chamada pelo Object Inspector para obter a representação do valor da propriedade em forma de string. Dentro desta função nós deveremos converter o valor da propriedade decimal para binário e depois transformar este valor em String, pois o Object Inspector só mostra Strings. Deste modo a função GetValue fica assim:**

```
unit BinaryPropEd;

interface

uses DsgnIntf;

type
  TBinIntegerProperty = class(TIntegerProperty)
  public
    function GetValue : string; override;
    procedure SetValue(const Value : String); override;
  end;

procedure Register;

implementation

Const
  Bits16 : Array [1..16] of Integer = (32768,16384,8192,4096,2048,1024,512,256,128,6

function TBinIntegerProperty.GetValue : string;
Var
  Num, i : integer;
begin
  Num:=GetOrdValue;
  Result := '0000000000000000';
  for i := 1 to 16 Do
    if ( Num >= Bits16[i] ) Then
    begin
      Num := Num - Bits16[i];
      Result[i] := '1';
    end;
  if ( Num > 0 ) Then
    raise EPropertyError.Create('Error converting '+IntToStr(GetOrdValue) + ' to bin
```

```
  Insert('B',Result,1);
end;

...

end.
```

Da implementação desta função a parte mais importante é a primeira linha:

```
Num:=GetOrdValue
```

Nós precisamos obter o valor que está naquele momento na propriedade para trabalhar nele. Porque nós usamos isto o método GetOrdValue, definido novamente em TPropertyEditor, o qual se encarrega de devolver o valor da propriedade em forma de ordinal (inteiro). De um modo semelhante, existem os métodos GetFloatValue, GetMethodValue, GetVarValue, etc. Para usar com o tipo de propriedade correspondente.

Uma vez armazenado o valor da propriedade na variável Num , a conversão começa do valor decimal para binário, o qual é fácil de entender. É bom saber que o número máximo de dígitos binários é 16, margem mais do que suficiente para a maioria das aplicações.

Por Ultimo nós temos que devolver um valor do tipo String como resultado da função. Porque nós vamos armazenar isto em uma propriedade que aceita somente String.
Para concluir, nós colocamos antes da cadeia de string uma letra ' B' para indicar que é base binária.

E isso é tudo que esta função faz. Deste modo, quando o Object Inspetor , ele chamará o método GetValue o qual lhe devolverá o string correspondente. Mas nós partimos para outro problema: seria bom se pudessemos introduzir o valor da propriedade tanto em decimal como em binário, não é mesmo?

Para adquirir esta funcionalidade nós devemos implementar (override) do método SetValue, definido na classe TPropertyEditor. Quando o usuário entra um valor novo no Object Inspetor, ele chama o método SetValue, o qual devera fazer a tradução inversa ao feito pelo método GetValue. Quer dizer, deveria converter o string que contém o valor novo da propriedade para o tipo de dados desta propriedade. Em nosso caso, o string entrará em base decimal ou binário (neste último caso, estará a primeira letra da cadeia ' B') e converter isto para um decimal. Porque nós implementaremos isto o método SetValue do modo seguinte:

```
...

type
  TBinIntegerProperty = class(TIntegerProperty)
  public
    function GetValue : string; override;
    procedure SetValue(const Value : String); override;
  end;

procedure Register;

implementation

...
```

```
procedure TBinIntegerProperty.SetValue(const Value : String);
Var
  i, Total, Longitud : integer;
   NumText : string;
begin
  if UpperCase(Value[1])='B' then
  begin
     NumText:=Copy(Trim(Value),2,Length(Trim(Value))-1);
     NumText:=Copy('0000000000000000',1,16-Length(NumText)) + NumText;
     Total:=0;
     for i:=1 to Length(NumText) do
     begin
       if not (NumText[i] in ['0','1']) then
         raise EPropertyError.Create(NumText[i] +' is not a valid binary digit')
       else if NumText[i]='1' then
         Total:=Total+Bits16[i];
     end;
     SetOrdValue(Total);
  end
  else
    SetOrdValue(StrToInt(Value));
end;

...

end.
```

Na implementação deste primeiro método conferimos nós se o usuário introduziu o valor novo da propriedade em base decimal ou em base binária. No primeiro caso, é só necessário converter o string para inteiro por meio da função StrToInt(Value) e, depois, usar o método SetOrdValue para armazenar o valor correspondente. De um modo semelhante, de acordo com o tipo da propriedade, existem os métodos SetFloatValue, etc.

No caso de a primeira letra da cadeia for um ' B', a cadeia se torna um valor binário para ser convertido em uma valor decimal, e usa o método SetOrdValue para armazenar o valor novamente na propriedade.

Uma vez implementado estes dois métodos (GetValue e SetValue) nós já temos nosso editor de propriedades acabado; nós só temos que registra-lo na VCL.

Inscrição de editor de propriedades

De mesmo modo que nós devemos registrar no VCL os componentes, nós temos que registrar os editores de propriedades tambem. Para isto temos o método RegisterPropertyEditor (definida na unidade DsgnIntf):

```
procedure RegisterPropertyEditor(PropertyType : PTypeInfo; ComponentClass : TClass;
                                                    const PropertyName : string; Ed
```

PropertyType faz referência ao tipo da propriedade para a qual o editor de propriedades será aplicado. Para dar um valor a este parâmetro nós normalmente usaremos a função TypeInfo, por exemplo, TypeInfo(integer).

ComponentClass permite restringir o uso do editor de propriedades para a classe específica. Um valor

nulo registra o editor para todos os componentes.

PropertyName especifica o nome da propriedade. Um valor diferente de nulo só registra o editor para a propriedade especificada, enquanto um valor ' ' registra isto para todas as propriedades

EditorClass especifica o editor de propriedades que registram (a classe).

Usando estes parâmetros, nós temos a nossa disposição um leque largo de possibilidades para registrar nosso editor de propriedades. Abaixo temos alguns exemplos com nosso editor recentemente criado:

- RegisterPropertyEditor(TypeInfo(integer), nil, '', TBinIntegerProperty) o editor de registros de propriedades para todos os componentes que têm uma propriedade de tipo inteiro. E é a forma mais global de registrar um componente e afeta a **todos** os componentes registrados no VCL. Se nós registramos nosso editor deste modo, nós veremos que todas as propriedades do tipo inteiro aparecem em binário! Nós também podemos introduzir um valor novo em decimal ou em binário (colocando a letra B). Nós estamos substituindo o editor de propriedades que Delphi usa para o nosso! :) Esta é a forma mais global de registrar editor de propriedades.
- RegisterPropertyEditor(TypeInfo (integer),TMiComponente,'PropriedadeBinaria',TBinIntegerProperty) o editor só registra e exclusivamente para a propriedade ' PropriedadeBinaria' do componente ' TMiComponente'. Esta é a forma restringida de registrar editor de propriedades.

Eu o aconselho que você registre o editor da forma mais global possivel para você experimentar-lo. Então, uma vez você viu todas as propriedades... você o desinstala.

Você também pode se criar um " componente de mentirinha " e registrar só o editor para o mesmo. Algo assim:

```
...

Type
  TMiComponente = class(TComponent)
   ...
     property PropriedadeBinaria : integer read FPropBin write FPropBin;
  ...
  end;
...
```

Localização de editor de propriedades

Como nós já mencionamos quando mostramos os passos que deveriam ser continuados criando editor de propriedades, o primeiro deles é criar uma unidade onde localizar o editor. Naquele momento dissemos nós que não uma escolha tão trivial como pudesse parecer em um pricincipio. Naquela unidade deveria se localizar o editor? na mesma unidade onde o componente que tem a propriedade que será publicada ?, em uma unidade separada? Nós temos três possíveis localizações:

A primeira opção é localizar o editor de propriedades na mesma unidade em que o componente. Esta é a opção mais intuitiva; porém não é o mais aconselhável, principalmente se o editor usa uma caixa diálogo.

A razão é que o Delphi só usa o editor de propriedades em design - time. De fato, a forma que

contém a caixa de diálogo não é conectada com a aplicação, nem o editor de classe de propriedades. Porém, se os recursos da caixa de dialogo são a unica coisa associados a unidade do componente então só estariamos aumentando o tamanho do executável. Então, nós estariamos aumentando o tamanho do executável para nada : (por outro lado, se o editor de propriedades não usa caixas de dialogo, continua ser aconselhavel.

A segunda opção é localizar o editor de propriedades (se usa uma forma ou uma caixa de diálogo) na mesma unidade da caixa diálogo. Deste modo, a aplicação que usa o componente não conecta ao editor de propriedades seus recursos associados.

A terceira opção é localizar o editor de propriedades em uma *unidade de registro. Uma unidade de registro é uma unidade normal e corrente que contém vários registros condenados correspondendo a componentes diferentes e editores de propriedades que residem em unidades diferentes. Esta é a opção mais aconselhável se o editor de propriedades é usado através de vários componentes.*

Então, as últimas duas opções são dependendo do caso, as mais aconselháveis.

Nas próximas unidades nós veremos exemplos de localizações diferentes dos editores de propriedades que nós iremos construir.

Codigo Fonte do editor de propriedades.

```
unit BinaryPropEd;

interface

uses DsgnIntf;

type
  TBinIntegerProperty = class(TIntegerProperty)
  public
    function GetValue : string; override;
    procedure SetValue(const Value : String); override;
  end;

procedure Register;

implementation

Const
  Bits16 : Array [1..16] of Integer = (32768,16384,8192,4096,2048,1024,512,256,128,6

function TBinIntegerProperty.GetValue : string;
Var
  Num, i : integer;
begin
  Num:=GetOrdValue;
  Result := '0000000000000000';
  for i := 1 to 16 Do
    if ( Num >= Bits16[i] ) Then
    begin
      Num := Num - Bits16[i];
      Result[i] := '1';
    end;
```

```
  if ( Num > 0 ) Then
    raise EPropertyError.Create('Error converting '+IntToStr(GetOrdValue) + ' to bin
  Insert('B',Result,1);
end;

procedure TBinIntegerProperty.SetValue(const Value : String);
Var
  i, Total, Longitud : integer;
  NumText : string;
begin
  if UpperCase(Value[1])='B' then
  begin
    NumText:=Copy(Trim(Value),2,Length(Trim(Value))-1);
    NumText:=Copy('0000000000000000',1,16-Length(NumText)) + NumText;
    Total:=0;
    for i:=1 to Length(NumText) do
    begin
      if not (NumText[i] in ['0','1']) then
        raise EPropertyError.Create(NumText[i] + ' is not a valid binary digit')
      else if NumText[i]='1' then
        Total:=Total+Bits16[i];
    end;
    SetOrdValue(Total);
  end
  else
    SetOrdValue(StrToInt(Value));
end;

procedure Register;
begin
  RegisterPropertyEditor(TypeInfo(Integer),nil,'',TBinIntegerProperty);
end;

end.
```

Luis Roche revueltaroche@redestb.es

Ultima modificación 24.08.1997