Adicionando Propriedades e Funcionalidades aos Componentes

Você já não precisou de uma determinada propriedade (que não foi implementada) num componente ? Por exemplo: Quem nunca precisou de uma propriedade “Alignment” no componente TEdit ? Ou ainda: Quem nunca exclamou: “Ah se este componente tivesse aquela propriedade (ou aquele evento, ou ainda, aquele método)...”.

Agora a pergunta é: Por que você mesmo não implementa esta propriedade (ou evento, ou método) ? É bem simples e é este o objetivo deste artigo.

O Problema

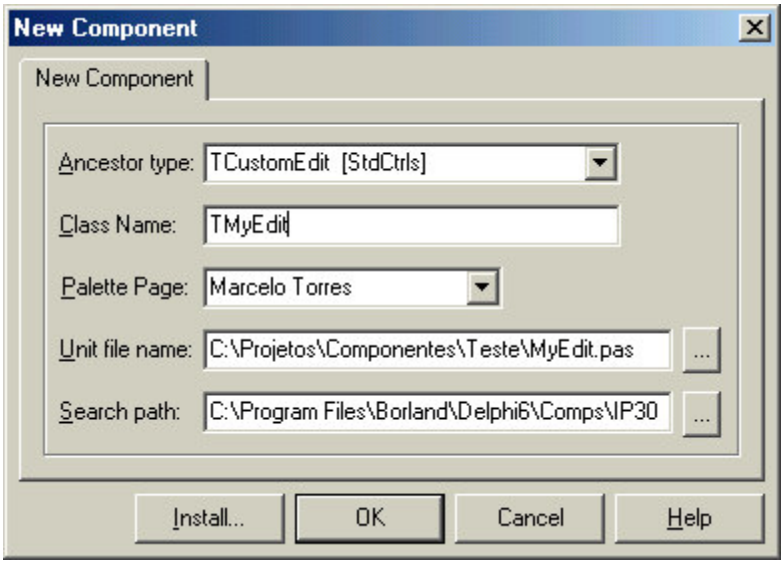
Digamos que precisemos de um componente TEdit que tenha as seguintes características a mais:

- Propriedade “Alignment” onde possamos alinhar o conteúdo de nosso TEdit a esquerda, a direita ou ao centro;
- Propriedade que possa armazenar uma cor para quando o TEdit obter o foco;
- Propriedade que possa informar o tipo de dado que está sendo digitado (string qualquer, CPF ou CNPJ);
- Caso o dado digitado seja um CPF ou CNPJ, um evento deve ser disparado caso os dígitos do CPF ou CNPJ sejam inválidos.

Acho que já temos “problemas” demais para um artigo só ! :-)

A Solução

A solução é simples. Primeiramente vamos desenvolver um novo componente. No Delphi basta clicar no menu “Component” e na opção “New Component...”. Vai abrir uma janela parecida com a figura abaixo:



No primeiro campo “Ancestor type” informamos de qual objeto iremos herdar as características. No nosso caso iremos herdar todas as características (propriedades, métodos e eventos) do TEdit. Como uma das propriedades que iremos herdar (propriedade de alinhamento) o TEdit não tem, é sabido que o objeto TCustomEdit possui. Como eu descobri isto é uma outra história que fica para outra ocasião.

No segundo campo “Class Name” informamos o nome do nosso objeto. No nosso caso coloquei TMyEdit (poderia ser qualquer outro conforme a conveniência).

No terceiro campo “Palette Page” informamos em qual paleta o novo componente será instalado. Pode ser uma previamente existente ou, no nosso caso, uma nova (com o singelo nome “Marcelo Torres”). :-)

No quarto campo “Unit file name” informamos o nome da unit que irá conter o nosso novo componente. Informe o nome do arquivo e o caminho onde o mesmo será gravado.

E, finalmente, no quinto campo “Search path” é especificado o caminho de onde o componente será encontrado pelos aplicativos que o contiverem. Normalmente nem é preciso alterar este campo que já vem previamente preenchido.

Clique no botão “OK” e a nova unit será aberta. A única coisa “automática” que veio é a classe de onde derivamos o nosso objeto (TCustomEdit no nosso caso) e a procedure Register que será a responsável por registrar o componente na paleta do Delphi.

Então vamos ao trabalho e implementar as nossas necessidades. Primeiro as mais fáceis para que nos sintamos encorajados para continuar. :-)

Vamos colocar a propriedade de alinhamento (Alignment).

Na seção “private” iremos colocar o seguinte:

```
private
  { Private declarations }
  FAlignment: TAlignment;
```

Esta variável (FAlignment) servirá para armazenarmos o conteúdo da propriedade “Alignment”.

Na seção “protected” colocaremos o seguinte:

```
protected
  { Protected declarations }
  procedure SetAlignment(const Value: TAlignment);
  procedure CreateParams(var Params: TCreateParams); override;
```

A procedure “SetAlignment” será a responsável por alimentar a nossa variável “FAlignment” e fazer um “refresh” no conteúdo do campo para que o mesmo respeite o alinhamento que especificamos.

A procedure “CreateParams” é a que realmente faz o alinhamento. Esta é uma procedure que já existe no objeto “TCustomEdit" e nós a herdamos para que possamos fazer as devidas alterações que forem necessárias.

Na seção “public” colocaremos o seguinte:

```
public
  { Public declarations }
  constructor Create(AOwner: TComponent); override;
```

Aqui nós estamos herdando mais uma procedure que é a responsável pela criação do objeto.

Na seção “published” iremos declarar todas as propriedades e eventos do TCustomEdit que queremos que esteja presente no nosso novo componente. Vai ficar assim:

```
published
  { Published declarations }
  property Alignment:TAlignment read FAlignment write SetAlignment;
  property AutoSelect;
  property AutoSize;
  property BorderStyle;
  property CharCase;
  property Color;
  property Ctl3D;
  property DragCursor;
  property DragMode;
  property Enabled;
  property Font;
  property HideSelection;
  property MaxLength;
  property OEMConvert;
  property ParentColor;
  property ParentCtl3D;
  property ParentFont;
  property ParentShowHint;
  property PasswordChar;
  property PopupMenu;
  property ReadOnly;
  property ShowHint;
  property TabOrder;
  property TabStop;
  property Text;
  property Visible;
  property OnChange;
  property OnClick;
  property OnDblClick;
  property OnDragDrop;
  property OnDragOver;
  property OnEndDrag;
  property OnEnter;
  property OnExit;
  property OnKeyDown;
  property OnKeyPress;
  property OnKeyUp;
  property OnMouseDown;
  property OnMouseMove;
  property OnMouseUp;
  property OnStartDrag;
```

Note que a primeira propriedade que colocamos (está em primeiro por causa da ordem alfabética) é a propriedade que iremos implementar “Alignment”.

Veja que a definição da mesma é:

```
    property Alignment:TAlignment read FAlignment write SetAlignment;
```

Declaramos o nome da propriedade (que vai aparecer no Object Inspector) “Alignment” o tipo desta propriedade “TAlignment” de qual variável vai ler o valor para alimentar a propriedade “FAlignment” e onde vai ser gravado o novo valor caso sofra alguma alteração. No nosso caso tem o nome de uma procedure, portanto, esta procedure vai ser executada quando o valor sofrer alguma alteração e caberá a ela gravar o valor alterado na variável “FAlignment”.

Até agora somente declaramos o protótipo do nosso novo componente. Passemos então para a parte de implementação.

Temos que criar a procedure que cria o componente.

```
constructor TMyEdit.Create(AOwner: TComponent);
begin
  inherited;
  FAlignment := taLeftJustify;
end;
```

A primeira linha desta procedure é a herança da procedure original. Logo após setamos a variável “FAlignment” para o valor default. Como esta procedure será executada apenas uma vez (quando colocamos o componente no formulário), aqui é o local ideal para inicializarmos todas as variáveis que precisam ser inicializadas.

Vamos criar também a procedure que seta a variável “FAlignment” quando houver alterações no seu valor.

```
procedure TMyEdit.SetAlignment(const Value: TAlignment);
begin
  if FAlignment <> Value then begin
     FAlignment := Value;
     RecreateWnd;
  end;
end;
```

Aqui o valor alterado da propriedade vem no parâmetro da procedure “Value”. Portanto, somente vamos fazer alguma coisa se o conteúdo deste parâmetro for diferente do que já estiver armazenado na variável “FAlignment”. Caso houver alguma alteração, além de colocar o novo valor na nossa variável, iremos chamar um método contido na classe WinControl que é a responsável por desenhar a tela (RecreateWnd).

Ao redesenhar a tela são utilizados algumas definições previamente estabelecidas para cada objeto, agora é hora de interceptar a criação destes parâmetros para que possamos colocar as características do alinhamento. É aí que entra a implementação da procedure “CreateParams”.

```
procedure TMyEdit.CreateParams(var Params: TCreateParams);
begin
  inherited CreateParams(Params);

  case Alignment of
    taLeftJustify:  Params.Style := Params.Style or LongWord(ES_Left);
    taRightJustify: Params.Style := Params.Style or LongWord(ES_Right);
    taCenter:       Params.Style := Params.Style or LongWord(ES_Center);
  end;
end;
```

Neste momento o nosso componente já é capaz de fazer alinhamento do seu conteúdo.

Mas vamos adiante !

O próximo passo agora é fazer com que o componente mude a cor quando receber o foco e volte a sua cor normal quando perder o foco.

Para isto devemos refazer os eventos OnEnter e OnExit, além de colocar uma nova propriedade para conter a cor de quando receber o foco.

Primeiramente devemos adicionar a unit “Graphics” na cláusula uses. Ela é a que tem a definição do tipo “TColor" que iremos precisar.

Na seção “Protected” devemos colocar duas procedures que são utilizadas para chamar os eventos OnEnter e OnExit (respectivamente).

```
    procedure DoEnter; override;
    procedure DoExit; override;
```

Vamos implementar estas procedures:

```
procedure TMyEdit.DoEnter;
begin
  Self.Color := FColorOnFocus;
  inherited;
end;

procedure TMyEdit.DoExit;
begin
  Self.Color := Color;
  inherited;
end;
```

Ao receber o foco, a cor é alterada para a cor que está na propriedade “ColorOnFocus” e quando perder o foco a cor é novamente alterada para a cor original (que está na propriedade “Color”).

Agora só falta colocar o valor default para esta propriedade. Isto é feito na criação do componente. Vamos alterar a procedure que faz isto adicionando a seguinte linha:

```
FColorOnFocus := clInfoBk;
```

Aqui foi definido que o cor default será o “InfoBk”. Caso queira outra, basta alterar esta linha colocando a cor desejada.
Neste ponto o nosso componente já é capaz de modificar sua cor quando recebe e quando perde o foco.

Vamos a mais uma característica. Uma propriedade que indica o conteúdo do nosso Edit que pode ser uma string qualquer, um CNPJ ou um CPF. Isto para formatar e validar estes dados internamente.

Para isto devemos definir um novo tipo, antes da definição do protótipo do nosso componente, logo abaixo da declaração “type”:

```
TData = (dtString, dtCNPJ, dtCPF);
```

Logo abaixo esta linha deve vir a linha de definição do nosso componente.

Na seção “private” devemos definir mais uma variável que irá armazenar o tipo de dado do nosso componente:

```
  FData: TData;
```

Devemos agora definir, na seção “private”, uma procedure que irá cuidar de alterar o valor da nossa variável caso a propriedade seja alterada:

```
  procedure SetData(Value: TData);
```

Também na seção “private” vamos colocar uma função para que seja formatada uma string para o caso de conter um CPF ou CNPJ. Isto significa considerar apenas números, formatar e setar a propriedade “Text” com o novo valor.

A linha que será adicionada na seção “private” é a seguinte:

```
  function FormataDado(Tipo: TData; Texto: String): String;
```

A seguir devemos fazer a implementação desta função. O código é o seguinte:

```
function TMyEdit.FormataDado(Tipo: TData; Texto: String): String;
var
  Num: String;
  Tam: Integer;
  Ind: Integer;

begin
  Result := '';
  if (Tipo <> dtCPF) and (Tipo <> dtCNPJ) then
     exit;

  Num := '';

  for Ind := 0 to Length(Texto) do
      if Pos(Copy(Texto, Ind, 1), '1234567890') > 0 then
         Num := Num + Copy(Texto, Ind, 1);

  if Num = '' then
     exit;

  if Tipo = dtCPF then
     Tam := 11
  else
     Tam := 14;

  Num := StringOfChar('0', Tam)+Num;
  Num := Copy(Num, Length(Num)-Tam+1, Tam);

  if Tipo = dtCPF then
     Num := Copy(Num, 1, 3) + '.' +
            Copy(Num, 4, 3) + '.' +
            Copy(Num, 7, 3) + '-' +
            Copy(Num, 10, 2)
  else
     Num := Copy(Num, 1, 2) + '.' +
            Copy(Num, 3, 3) + '.' +
            Copy(Num, 6, 3) + '/' +
            Copy(Num, 9, 4) + '-' +
            Copy(Num, 12, 2);

  Result := Num;
end;
```

Já temos uma propriedade que identifica o tipo de dado que está sendo digitado no nosso Edit, já temos uma função capaz de formatar um CPF ou um CNPJ. Falta agora fazer com que quando o componente perder o foco todo o conteúdo seja formatado levando em consideração o tipo de dado que foi informado na propriedade "Data". Isto pode ser feito alterando a procedure responsável pela perda do foco (DoExit).

Isto é feito adicionando o seguinte na procedure "DoExit":

```
  if (FData = dtCPF) or (FData = dtCNPJ) then
     Text := FormataDado(FData, Text);
```

Importante: Isto deve ser inserido antes do comando "inherited".

Agora é hora de alterarmos a procedure "SetData". Esta é a procedure responsável por armazenar o novo valor da nossa variável "FData" (que especifica o tipo de dados que será digitado pelo usuário).

Esta procedure ficará assim:

```
procedure TMyEdit.SetData(Value: TData);
begin
  if Value <> FData then begin
     FData := Value;

     if (FData = dtCNPJ) or (FData = dtCPF) then
        Text  := FormataDado(FData, Text);
  end;
end;
```

Se o tipo de dado for um CPF ou um CNPJ temos que formatar o texto que está na propriedade "Text".

Já que estamos formatando o CPF ou o CNPJ, nada mais justo que o validemos. Caso haja algum erro nos dígitos verificadores disparar um evento indicando que houve algum erro.

Para isto vai ser necessário uma nova função (que valida o CPF ou CNPJ) e um novo evento no nosso componente indicando o erro que aconteceu.

Vamos por partes. Primeiro a função que verifica os dígitos verificadores do CPF e do CNPJ. Agradecimentos especiais ao amigo ***Adenilton Rodrigues*** pela função. Eu tinha duas funções, uma para o CPF e outra para o CNPJ. Este nosso amigo tem apenas uma. Otimização sempre é bem vinda...

Sem mais delongas, vamos ao código:

```
function TMyEdit.VerCPFCNPJ(Text: String): String;
var
  Cnt:  Integer;
  Ind:  Integer;
  Mlt:  Integer;
  Som:  Integer;
  Dig:  Integer;
  Num:  String;

begin
  Num := '';
  for Ind := 1 to Length(Text) do
      if Pos(Copy(Text, Ind, 1), '1234567890') > 0 then
         Num := Num + Copy(Text, Ind, 1);

  CNPJ   := (Length(Num)=12);
  Result := Num;

  for Cnt := 1 to 2 do begin
    Mlt := 2;
    Som := 0;

    for Ind := Length(Result) downto 1 do begin
       Som := Som + (Ord(Result[Ind])-Ord('0')) * Mlt;

       Inc(Mlt);
       if (Mlt > 9) and CNPJ then
          Mlt := 2;
    end;

    Dig := 11 - Som mod 11;

    if Dig >= 10 then
       Dig := 0;

    Result := Result + Chr(Dig + Ord('0'));
  end;

  Result := Copy(Result, Length(Result)-1, 2);
end;
```

Bem, não vou perder tempo (e espaço) explicando como os dígitos verificadores do CPF e do CNPJ são calculados. Além do mais acho que o editor desta revista não ia gostar muito. :-) Mas a função que calcula estes dígitos (tanto para CPF quanto para o CNPJ) está aí em cima...

Não podemos nos esquecer de colocar a declaração desta função na seção “Private” do componente. Coloque logo abaixo da declaração da função “FormataDado” o seguinte:

```
    function VerCPFCNPJ(Text: String): String;
```

Para adicionarmos um novo evento, antes é necessário declará-lo. Logo abaixo da definição do tipo “Data” vamos colocar a definição do novo evento:

```
  TerrorEvent = procedure(Sender: TObject;
                          Data: TData;
                          Msg: String) of object;
```

Temos também que criar uma variável para armazenar o conteúdo do nosso evento. Basta adicionar a seguinte linha na seção “Private” junto das declarações das variáveis:

```
    FOnError:   TErrorEvent;
```

Agora é hora de publicarmos o evento na seção “Published”. É só colocar a seguinte linha:

```
    property OnError: TErrorEvent read FOnError write FOnError;
```

Este evento será disparado quando o componente perder o foco e o tipo de dado for um CPF ou um CNPJ e os dígitos verificadores estiverem inválidos. Traduzindo isto em programação significa que devemos alterar a procedure “DoExit”. Esta procedure alterada fica assim:

```
procedure TMyEdit.DoExit;
var
  Dig:  String;
  Calc: String;

begin
  Text := FormataDado(FData, Text);

  Self.Color := Color;

  if Assigned(FOnError) then begin
     if (FData = dtCNPJ) or (FData = dtCPF) then begin
        Dig  := Copy(Text, Length(Text)-1, 2);
        Calc := VerCPFCNPJ(Copy(Text, 1, Length(Text)-2));

        if Dig <> Calc then
           if FData = dtCNPJ then
              FOnError(Self, FData, 'CNPJ Inválido')
           else
              FOnError(Self, FData, 'CPF Inválido');
     end;
  end;

  inherited;
end;
```

Com isso, se o CPF ou CNPJ estiverem com os dígitos verificadores inválidos este evento será disparado passando o tipo de dado (CPF ou CNPJ) e a mensagem de erro.

A nossa proposta foi cumprida. Pele menos em termos de programação. Segue aqui o código completo do componente:

```
unit MyEdit;

interface

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

type
  TData = (dtString, dtCNPJ, dtCPF);
  TErrorEvent = procedure(Sender: TObject;
                          Data: TData;
                          Msg: String) of object;

  TMyEdit = class(TCustomEdit)
  private
    { Private declarations }
    FAlignment: TAlignment;
    FColorOnFocus: TColor;
    FData: TData;

    FOnError:   TErrorEvent;

    procedure SetData(Value: TData);
    function FormataDado(Tipo: TData; Texto: String): String;
    function VerCPFCNPJ(Text: String): String;
  protected
    { Protected declarations }
    procedure CreateParams(var Params: TCreateParams); override;
    procedure SetAlignment(const Value: TAlignment);
    procedure DoEnter; override;
    procedure DoExit; override;
  public
    { Public declarations }
    constructor Create(AOwner: TComponent); override;
  published
    { Published declarations }
    property Alignment: TAlignment read FAlignment write SetAlignment;
    property AutoSelect;
    property AutoSize;
    property BorderStyle;
    property CharCase;
    property Color;
    property ColorOnFocus: TColor read FColorOnFocus write FColorOnFocus;
    property Ctl3D;
    property Data: TData read FData write SetData;
    property DragCursor;
    property DragMode;
    property Enabled;
    property Font;
    property HideSelection;
    property MaxLength;
    property OEMConvert;
    property ParentColor;
    property ParentCtl3D;
```

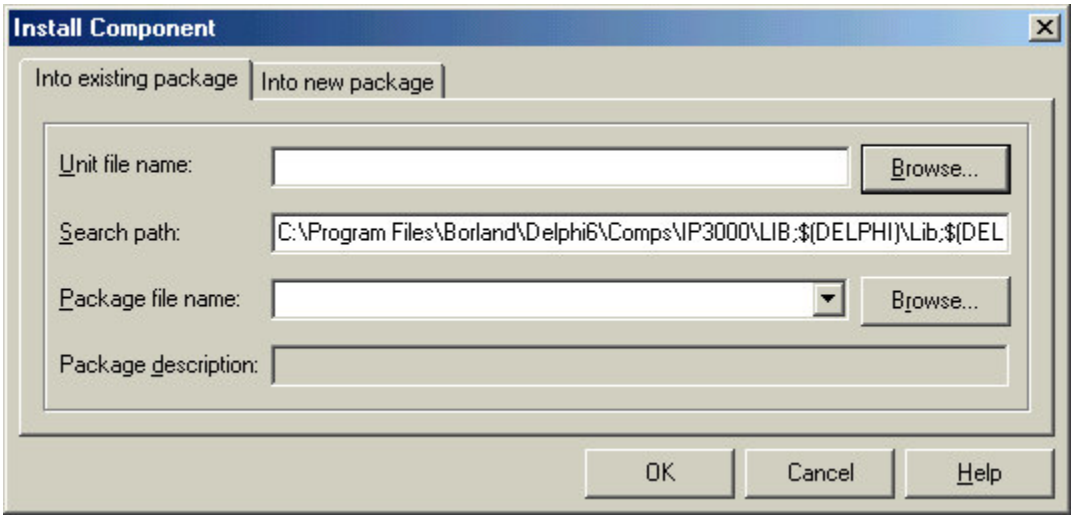
```
    property ParentFont;
    property ParentShowHint;
    property PasswordChar;
    property PopupMenu;
    property ReadOnly;
    property ShowHint;
    property TabOrder;
    property TabStop;
    property Text;
    property Visible;
    property OnChange;
    property OnClick;
    property OnDblClick;
    property OnDragDrop;
    property OnDragOver;
    property OnEndDrag;
    property OnEnter;
    property OnError: TErrorEvent read FOnError write FOnError;
    property OnExit;
    property OnKeyDown;
    property OnKeyPress;
    property OnKeyUp;
    property OnMouseDown;
    property OnMouseMove;
    property OnMouseUp;
    property OnStartDrag;
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Standard', [TMyEdit]);
end;

constructor TMyEdit.Create(AOwner: TComponent);
begin
  inherited;

  FAlignment := taLeftJustify;
  FColorOnFocus := clInfoBk;
end;

procedure TMyEdit.CreateParams(var Params: TCreateParams);
begin
  inherited CreateParams(Params);

  case Alignment of
    taLeftJustify:  Params.Style := Params.Style or LongWord(ES_Left);
    taRightJustify: Params.Style := Params.Style or LongWord(ES_Right);
    taCenter:       Params.Style := Params.Style or LongWord(ES_Center);
  end;
end;
```

```
procedure TMyEdit.SetAlignment(const Value: TAlignment);
begin
  if FAlignment <> Value then begin
     FAlignment := Value;
     RecreateWnd;
  end;
end;

procedure TMyEdit.DoEnter;
begin
  Self.Color := FColorOnFocus;
  inherited;
end;

procedure TMyEdit.DoExit;
var
  Dig:  String;
  Calc: String;

begin
  Text := FormataDado(FData, Text);

  Self.Color := Color;

  if Assigned(FOnError) then begin
     if (FData = dtCNPJ) or (FData = dtCPF) then begin
        Dig  := Copy(Text, Length(Text)-1, 2);
        Calc := VerCPFCNPJ(Copy(Text, 1, Length(Text)-2));

        if Dig <> Calc then
           if FData = dtCNPJ then
              FOnError(Self, FData, 'CNPJ Inválido')
           else
              FOnError(Self, FData, 'CPF Inválido');
     end;
  end;

  inherited;
end;

procedure TMyEdit.SetData(Value: TData);
begin
  if Value <> FData then begin
     FData := Value;

     if (FData = dtCPF) or (FData = dtCNPJ) then
        Text  := FormataDado(FData, Text);
  end;
end;

function TMyEdit.FormataDado(Tipo: TData; Texto: String): String;
var
  Num: String;
  Tam: Integer;
  Ind: Integer;

begin
```

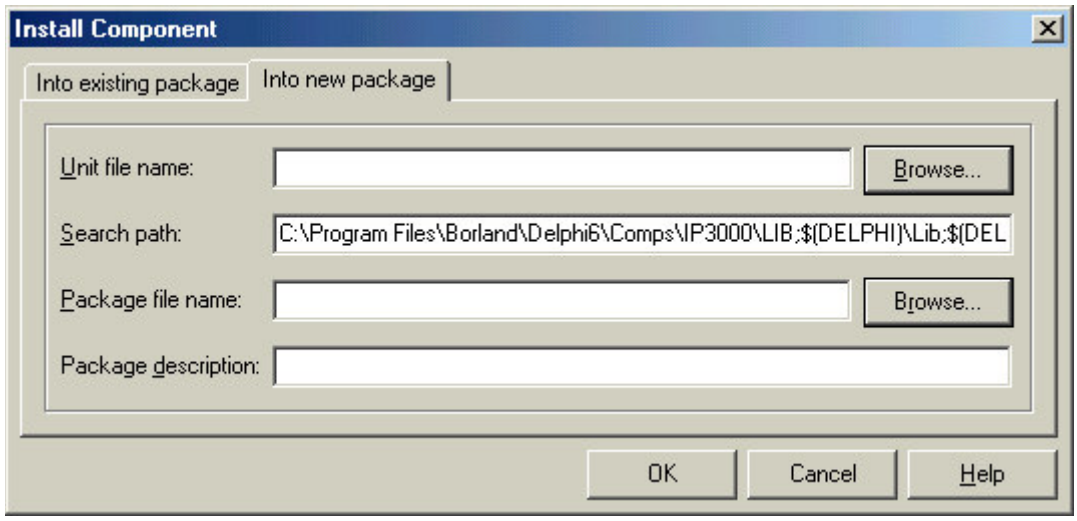
```
  Result := '';
  if (Tipo <> dtCPF) and (Tipo <> dtCNPJ) then
     exit;

  Num := '';

  for Ind := 0 to Length(Texto) do
      if Pos(Copy(Texto, Ind, 1), '1234567890') > 0 then
         Num := Num + Copy(Texto, Ind, 1);

  if Num = '' then
     exit;

  if Tipo = dtCPF then
     Tam := 11
  else
     Tam := 14;

  Num := StringOfChar('0', Tam)+Num;
  Num := Copy(Num, Length(Num)-Tam+1, Tam);

  if Tipo = dtCPF then
     Num := Copy(Num, 1, 3) + '.' +
            Copy(Num, 4, 3) + '.' +
            Copy(Num, 7, 3) + '-' +
            Copy(Num, 10, 2)
  else
     Num := Copy(Num, 1, 2) + '.' +
            Copy(Num, 3, 3) + '.' +
            Copy(Num, 6, 3) + '/' +
            Copy(Num, 9, 4) + '-' +
            Copy(Num, 12, 2);

  Result := Num;
end;

function TMyEdit.VerCPFCNPJ(Text: String): String;
var
  Cnt:  Integer;
  Ind:  Integer;
  Mlt:  Integer;
  Som:  Integer;
  Dig:  Integer;
  Cnpj: Boolean;
  Num:  String;

begin
  Num := '';
  for Ind := 1 to Length(Text) do
      if Pos(Copy(Text, Ind, 1), '1234567890') > 0 then
         Num := Num + Copy(Text, Ind, 1);

  CNPJ   := (Length(Num)=12);
  Result := Num;

  for Cnt := 1 to 2 do begin
    Mlt := 2;
```

```
    Som := 0;

    for Ind := Length(Result) downto 1 do begin
       Som := Som + (Ord(Result[Ind])-Ord('0')) * Mlt;

       Inc(Mlt);
       if (Mlt > 9) and CNPJ then
          Mlt := 2;
    end;

    Dig := 11 - Som mod 11;

    if Dig >= 10 then
       Dig := 0;

    Result := Result + Chr(Dig + Ord('0'));
  end;

  Result := Copy(Result, Length(Result)-1, 2);
end;

end.
```

Bem. O componente é isso aí. Para instalá-lo basta clicar no menu “Component” e depois na opção “Install Component”. Quando aparecer a janela abaixo, clique na guia “Into new package”.



Ao clicar na gua “Into new package” irá aparecer uma tela como esta:



No primeiro campo é necessário informar o nome da unit que contém o componente. Incluindo o seu caminho.

No segundo campo (que já vem preenchido) são as pastas onde ele vai buscar outras units caso sejam necessárias. Não precisa mexer neste campo.

No terceiro campo é o nome do “Package” que será criado. Recomenda-se colocar um nome diferente da unit para evitar conflitos.

No quarto campo é só uma descrição do componente.

Após informar os campos desta tela e clicar no botão OK, vai abrir uma tela com toda a unit do componente e logo em seguida abre uma janela perguntando se deseja realmente instalar o componente. Clique no botão “Yes”.

Vai abrir outra tela com o package já criado. Clique no botão “Compile” e logo depois no botão “Install”.

Se tudo correr bem, o componente estará instalado.

Falta ainda colocar um ícone para ele, mas isto é papo para uma outra vez.

Um abraço,
Marcelo Torres