# Curso de criação de componentes em Delphi

## Unidade 4. Adicionando recursos a um componente existente: TBlinkLabel

Voltar ao índice

*Por Luis Roche*

---

Nesta unidade nós aprenderemos como já adicionar recursos e possibilidades para um componente Delphi existente. Nós aprenderemos a usar objetos (componentes) em nosso próprio componente, o que eles são e como se declaram os constructors e os destructors.

Tudo isso,claro é, baseado em um exemplo concreto: TBlinkLabel.

## Objetivo do componente

Nosso propósito é criar um componente idêntico ao o tipo TLabel , mas com uma capacidade a mais: o de piscar. Quer dizer, nosso componente deveria nos mostrar a mensagem introduzida na de propriedade que pisca com uma certa freqüência. Também, se o valor introduzido freqüência é nulo, a mensagem deverá permanecer fixa na tela (emulando o que acontece a um Tlabel normal).

## Usando um componente existente em nosso próprio componente. Cronômetros.

Como nós há pouco dissemos, nosso componente deveria piscar com uma certa freqüência. Porque nós precisamos ser possíveis medir o tempo que devera piscar a mensagem. Afortunadamente, o Delphi nos provê um objeto que molda perfeitamente a nossas exigências de medição de tempo: TTimer.

Embora não é este o lugar de explicar as características do Cronômetro de objeto (a informação básica deste componente pode estar na própria ajuda on-line do Delphi), ele serve para parar um momento na propriedade do cronômetro que nós usaremos: o *intervalo da* propriedade. *Esta propriedade nos permite especificar o intervalo de tempo que nós queremos medir (em milisegundos). Quando o valor estabelecido é alcançado o método que esta especificado no evento OnTimer é executado.*

*O método que nós especificaremos para o evento OnTimer fara com que a mensagem apareca e*

*desapareça.*

*A teoria é muito simples, mas... como nós adicionamos o objeto cronômetro para nosso componente? Em design-time é simples. É só selecionar o cronômetro na paleta de componentes e colocá-lo em nossa forma.*

*Mas quando estamos escrevendo um componente nós não podemos usar este sistema, se nós não vamos isto à mão. Os passos gerais que devem ser seguidos:*

- *Adicionar a unidade que o componente é definido na claúsula uses de nossa unidade. Como o objeto cronômetro é declarado na unidade ExtCtrls , é esta unidade que o que nós deveríamos somar ao claúsula uses.*
- *Adicionar o método necessário para construir o objeto (em nosso caso, o cronômetro).*
- *Escrever o código necessário para destruir o objeto no método destructor de nosso componente.*

*Na seção seguinte nós aprofundaremos nestes dois passos.*

# Construtores e Destruidores.

## *Construtores*

Os objetos que nós declaramos em um componente não existem em memória até que objeto é criado (também é dito que uma instância do objeto é criada) para uma chamada para o método de construção do objeto.

Um Construtor não é mais que um método que provê memória para o objeto e aponta (por meio de um ponteiro) para ele. Chamando ao método **create**, o construtor nomeia a instância do objeto para uma variável.

Ele serve para fazer advertência que todos os componentes herdam um método denominado create que se encarrega de criar o componente em memória. De um modo adicional, no método create os valores de certos campos do componente podem ser inicializados (comumente lhes nomear um valor por padrão)

No método create do componente nós podemos criar objetos adicionais que ele precisa para nosso componente. Em nosso caso, no método create de TBlinkLbl nós criaremos o cronômetro.

## *Destruidores*

Quando nós terminamos de usar um objeto, nós deveríamos destruir isto, quer dizer, liberar a memória que ocupou o objeto. Esta operação é feita atraves do método **destroy** que todos os componentes herdam de TComponent.

Se nós criamos algum objeto adicional, nós deveríamos destruir também escrevendo isto noo método destroy.

Delphi cria e destrói nosso componente automaticamente quando é necessário, como ja foi dito, os métodos create e destroy são herdados do TComponent.

Mas se nós queremos usar algum objeto em nosso componente, o Delphi não cria e destroi

automaticamente, nós devemos fazer isto manualmente. Este processo é feito para adicionar o código necessário para os métodos  create e destroy. É importante dizer que nós escreveremos o mesmo código novamente no componente, e podem sobrecarregar o código novo.

De um modo geral isto é feito do seguinte modo: (um exemplo mais concreto; isto tem na próprio código fonte do componente)

- Na parte pública de nosso componente nós declaramos os métodos o create e destroy seguido pela palavra chave overrride:

```
public
                constructor create(AOwner : TComponent); override;
                destructor   destroy; override;
```

- Escrever o código adicional do construtor e do destruidor na seção de implementação.

```
constructor TComponent.Create(AOwner : TComponent);
     begin
          inherited Create(AOwner);
          ...
     end;

     destructor TComponent.Destroy;
     begin
          ...
          inherited destroy;
     end;
```

  A coisa mais importante para se lembrar de é que quando sobrecarregamos um construtor, a primeira coisa que deveria ser feita é uma ligação ao construtor original (linha herdada o create). Próximo a pessoa pode somar o código necessário para criar o objeto.

  Da mesma maneira, quando sobrecarregamos um destruidor, os objetos criados antes deveriam ser liberados e a última linha deveria ser uma chamada ao destruidor original.

# Implememtando o piscador.

Em design-time do componente nós deixamos ainda um passo importante: como fazer as piscadas de mensagem. Nós dissemos que quando o evento acontece OnTimer o método será executado e mostrará e ocultara a mensagem dando uma sensação de "pisca-pisca"

A solução é muito simples, desde que nosso componente, descendendo de um TLabel uma propriedade que determina se o componente deveria ser visível ou não. Esta propriedade é denominada **visible.**

**Deste modo, nosso método pisca alternará o valor da propriedade booleana visible quando o evento OnTimer acontece. São mostrados os detalhes concretos de implementação no e código fonte.**

# Outros detalhes na criação do componente. Valores padrão.

Desta forma,  nós concluimos dois aspectos na criação do componente:

- Nós comentamos que se o valor introduzido na propriedade freqüência for nulo, a mensagem deveria permanecer fixa em tela. Isto é alcançado ativando e incapacitando o componente quando é introduzido o valor da velocidade de propriedade no Object Inspector (ver o procedimento SetVelocidad no código fonte).
  Deste modo, se é introduzido 0 como valor da velocidade, o cronômetro é simplesmente inválido e ele usa o valor da propriedade visível para True. Se o valor introduzido é diferente de zero, o cronômetro é habilitado e, por conseguinte, o pisca - pisca.
- Um último detalhe é dar um valor padrão para a propriedade que define a velocidade. Isto é adquirido em dois passos: primeiramente na declaração da propriedade velocidade a palavra chave **Default** seguida pelo valor é somada por padrão (400 em nosso caso). Logo que o construtor é inicializado o campo associado com a propriedade (FVelocidad:=400). Os **dois passos são necessários e não** são possíveis não obviar nenhum de ambos.
  Este detalhe de nomear valores padrão é mais importante para evitarmos erros de inicialização e temos certeza de que a propriedade sempre terá um valor válido. O Delphi só manterá o valor da propriedade se ele for diferente ao valor padrão.

# O Código Fonte do componente

```
unit Blinklbl;      { (c)1996 by Luis Roche }

interface

uses
  Classes, StdCtrls, ExtCtrls;
type
  TBlinkLabel = class(TLabel)

  private
    FVelocidad : integer;

    FTimer : TTimer;
    procedure SetVelocidad(valor : integer);
  protected
    procedure parpadea(Sender : TObject);
  public
    constructor Create(AOwner : TComponent); override;    {Constructor}
    destructor Destroy; override;                         {Destructor}
  published
    property Velocidad : integer read FVelocidad write SetVelocidad default 400;
  end;

procedure Register;

implementation

constructor TBlinkLabel.Create(AOwner : TComponent);
begin
  inherited Create(AOwner);       {Chama o constructor original (herdado)}
  FTimer := TTimer.Create(Self);  {Criamos o timer}
  FVelocidad := 400;              {Freqüência (velocidade) padrão}
  FTimer.Enabled:=True;           {Ativamos o timer}
  FTimer.OnTimer:=parpadea;       {Pegamos o método parpadea (Pisca-Pisca)}
  FTimer.Interval:=FVelocidad;    {Pegamos o intervalo do timer = freqüência parpade
end;
```

```
destructor TBlinkLabel.Destroy;
begin
  FTimer.Free;          {Liberamos o timer}
  inherited destroy;    {Chamamos o destructor original (herdado)}
end;

procedure TBlinkLabel.SetVelocidad (valor : integer);
begin
  If FVelocidad <> valor then        {Só se o valor introduzido é diferente do padrã
  begin
    if valor < 0 then FVelocidad:=0;
    FVelocidad:=Valor;               {Aceita a velocidade}
    if FVelocidad=0 then FTimer.Enabled:=False else FTimer.Enabled:=True;
    {Se Fvelocidad=0 a menssagem deve estar sempre visible}
    FTimer.Interval:=FVelocidad;
    Visible:=True;
  end;
end;

procedure TBlinkLabel.parpadea(Sender : TObject);
begin
  if FTimer.Enabled then Visible := not(Visible);  {Alternativamente mostra e oculta
end;

procedure Register;        {Registro do componente}
begin
  RegisterComponents('Curso', [TBlinkLabel]);
end;

end.
```

---

*Luis Roche* *revueltaroche@redestb.es*

Ultima modificación 9.12.1996