Curso de criação de componentes em Delphi

Unidade 6. TMultiGrid: cores, alinhamento e linhas múltiplas em um TStringGrid.

Voltar ao índice

Por Luis Roche

Nesta unidade nós criaremos um componente melhorado do tipo StringGrid. Nós aprenderemos a criar eventos que nos permitirão dotar de possibilidades novas e potentes a nossos componentes. Também, nós estudaremos o método que OnDrawCell e já nos aprofundaremos nos tratados de tópicos relacionando unidades prévias para os objetos Canvas e Brush.

Objetivo do componente

Como nós há pouco mencionamos, nosso propósito é criar um componente do tipo StringGrid mas com funcionalidades novas. De forma que a primeira coisa que nós deveremos saber é o que o componente TStrinGrid padrão nos permite fazer e o que não nos permite. Assim se você ainda não conhece este objeto, nós faremos uma pequena pausa para vc parar e olhar na ajuda online do Delphi. Já está? Bem, porque agora as características novas que nós implementaremos a nosso componente: a cor e o alinhamento das celulas ao nível que nós queremos (coluna, linhas e até mesmo celulas individuais, incluindo alinhamento vertical) como também uma propriedade nova denominada multilinha que nos permitirá mostrar mais que uma linha de texto em cada celula do grid .

A figura que abaixo mostra um exemplo do componente em operação:

	Enero	Febrero	Total Año	Notas
Zona A	1.000.000	1.250.000	9.150.000	Incremento sobre año anterior
Zona B	1.450.000	950.000	4.150.000	Decremento
Resto de Zonas	4.000.000	3.250.000	17.250.000	Incremento sobre año anterior
TOTAL	6.450.000	5.450.000	30.550.000	

Implementando o alinhamento de celulas. Criando nossos próprios eventos.

Nós começaremos com a propriedade de alinhamento. A primeira coisa que nós deveremos decidir é como nós implantaremos isto. Fundamentalmente, há três possibilidades:

> Definindo o alinhamento de um modo global. Se você optar por esta implementação, você apenas teria que definir uma propriedade (denominada, logicamente, Alignment) isso controlaria o alinhamento de todas as celulas do grid. Isto seria muito simples, mas muito não muito prático, desde que o que queremos ter são: as celulas de cabeçalho centralizadas, textos alinhados à esquerda, ou números à direita, etc.
>
> Uma solução um pouco melhor: definir o alinhamento a nível de colunas. Deste modo cada coluna pode ser alinhada com independênte da outra, mas mesmo assim todas as linhas desta coluna terá o mesmo alinhamento (cabeçalho e dados). Se vc optar por esta solução você terá um problema na sua implementação pois deveríamos criar um editor de propriedades, e nós ainda não sabemos como fazer isto (paciência, que será visto nas próximas unidades). Este é o caso do editor de colunas que Delphi 2 incorpora.
>
> A terceira possibilidade nos oferece um controle total: definição do alinhamento de cada celula a nível individual. Deste modo cada celula terá seu próprio alinhamento com independente do outro.O problema é que isto requer um pouco mais que esforço por parte do usuário do componente.

Como você vê , cada uma das três opções acima tem suas vantagens e desvantagens. Por isso em nosso componente nós implementaremos uma combinação do primeiro e terceiro método. Deste modo, nós definiremos uma propriedade de Alinhamento específica para o alinhamento individual por celula e outro padrao para alinhamento total da grade.

A implementação do alinhamento horizontal a nível global não tem nenhum mistério: Basta definir o propriedade Alignment do tipo TAlignment (já incluida no Delphi). O campo que ela manterá o valor desta propriedade terá o nome: FAlignment. Para escrevermos o valor na propriedade, usaremos o método SetAlignment, e a leitura sera feita diretamente no campo FAlignment. Deste modo nós definimos a interface da propriedade Alignment perfeitamente. Nós definiremos as propriedades para o alinhamento individual por celula quando estudarmos o evento OnDrawCell.

O alinhamento vertical é desenvolvido de um modo semelhante. O único aspecto diferencial é que o Delphi não incorpora a propriedade do tipo TVerticalAlignment, de forma que nós deveremos cria-lo:

```
TVerticalAlignment = (vaTop, vaCenter, vaBottom);
```

Nós veremos agora como implementar a interface do alinhamento das celulas a nível individual. Para isso nós criaremos um novo **evento**, que ao ser ativado saberemos o alinhamento de uma celula qualquer.

Como já você viu na unidade 2, um evento é um mecanismo que víncula uma ação para certo código. Mais concretamente, um evento é um ponteiro ao método.

A implementação do evento é feito por meio de propriedades, quer dizer, os eventos são propriedades. Ao contrário das propriedades padrões, os eventos não usam métodos para implementar as partes lidas e escritas. No lugar delas, as propriedades de evento utilizam um campo privado.

Mas já temos bastante de teoria, vamos trabalhar!. Como já foi mencionado, nós criaremos um evento novo que deverá ser ativado toda vez que precisarmos obter o valor do alinhamento de uma celula específica. A primeira coisa que nós devemos fazer então, é definir nosso tipo de evento. Isto se faz por meio seguinte linha:

```
TMultiGridAlignmentEvent=procedure(Sender:TObject; ARow,ACol:LongInt;
  var HorAlignment: TAlignment; var VerAlignment: TVerticalAlignment) of object;
```

Os parâmetros do evento TMultiGridAlignmentEvent:

- Sender: Identifica o objeto que faz a ligação.
- ARow e ACol: Identificam as coordenadas de Linha(ARow) e Coluna (ACol) para o alinhamento.

  var HorAlignment e var VerAlignment: Parâmetros para alinhamento horizontal (HorAlignment) e vertical (VerAlignment).

Nós já definimos o tipo de evento. Agora nós deveremos criar um campo que contenha o estado da propriedade OnGetCellAlignment. Isto devera ser feito na parte Private:

```
private
   ...
   FOnGetCellAlignment: TMultiGridAlignmentEvent;
```

Finalmente, nós definiremos a propriedade na parte Published:

```
  property OnGetCellAlignment: TMultiGridAlignmentEvent read FOnGetCellAlignment w
```

Nós só ativaremos o evento quando precisarmos, embora um pouco mais tarde faremos com mais detalhes em nosso componente, a linha abaixo nos mostra um mecanismo geral:

```
if Assigned(FOnGetCellAlignment) then
  FOnGetCellAlignment(Self,ARow,ACol,HorAlignment,VerAlignment);
```

Importante: Antes de ativar um evento, é conveniente olhar primeiro se este evento tem um gerenciador de evento nomeado, como o usuário do componente não tem porque ter escrito este gerenciador. De lá a comparação *if Assigned:* se há um gerenciador evento, ele é chamado, mas se não

há nada é feito.

Implementando a fonte, estilo e cor de celulas.

Se você entendeu a seção anterior, não haverá nenhum problema para entender esta, como a forma de implementar a cor e a atribuição de fonte em uma certa celula será feita de um modo semelhante.

Defininimos um novo evento que será ativado quando é necessário determinar os atributos da celula. Para isto, nós criaremos o tipo do evento, o campo privado armazenará isto e a propriedade associada a este evento:

```
TMultiGridColorEvent=procedure(Sender: TObject; ARow,Acol: LongInt;

   AState: TGridDrawState; Abrush: TBrush; AFont:TFont) of object;

...

FOnGetCellColor: TMultiGridColorEvent;

...

property OnGetCellColor: TMultiGridColorEvent read FOnGetCellColor write FOnGetCel
```

A diferença principal entre este evento e o correspondente ao alinhamento esta determinado pelos parâmetros ABruh e AFont. O usuário do componente devera retornar o brush e a Fonte a celula correspondente nas coordenadas ARow e ACol. O parâmetro AState nos informa do estado da cela (selected, focus, fixo...)

Celulas Multi-linhas.

Vamos passar agora para a implementação das celulas multi-linhas.

Em primeiro lugar nós definiremos a interface. Para isto, nós criaremos uma propriedade nova chamada MultiLinha . Esta propriedade será armazenada no campo FMultiLinha que será do tipo boolean. Se FMultiLinha é false, nosso componente se comportará como o StringGrid normal, enquanto se for true, se tornará um StringGrid Multi-Linhas.

```
private

  FMultiLinha: Boolean;

...

  property MultiLinha: Boolean read FMultiLinha write SetMultiLinha default False;
```

Ele serve para fazer uma chamada ao método SetMultiLinha, se um valor novo é colocada para FMultiLinha o componente é redesenhado por meio da instrução o Invalidate. Esta mesma técnica é usada nos métodos SetAlignment, SetVerticalAlignment e SetColor.

O coração do componente: o método DrawCell.

Até agora, nós nos concentramos na interface de nosso componente; chegou o momento de definir o implementação. O processo inteiro de desenho de uma celula é feita através do método DrawCell. Este método é o verdadeiro coração de nosso componente, visto que é nele que deve ser escrito todo o código de calculo e desenho do texto correspondente em uma determinada celula. O evento OnDrawCell passa os seguintes parâmetros ao método DrawCell:

- ACol e ARow: Coordenadas da celula.
- ARect: Estrutura de tipo retangular que identifica o canto superior esquerdo e o direito inferior (em pixels) da celula desejada.
- AState: É o estado atual da celula (selecionda, enfocada ou fixa). Em princípio nós não usaremos o último valor neste parâmetro.

Nós já sabemos onde. Agora falta ver como é. A princípio você pode ficar assustado com tudo aquilo nós temos que fazer: calcular o alinhamento horizontal e vertical, ativar os eventos de alinhamento e colorir, fragmenar o texto de uma celula em várias linhas... Mas não há nenhuma razão: o Delphi e as API do Windows nos ajudam a diminuir tudo isso para 20 ou 30 linhas compreensiveis. Logo o código que corresponde ao método DrawCell mostrado:

```
procedure TMultiGrid.DrawCell(ACol,ARow : LongInt; ARect : TRect; AState : TGridDraw
Const
  TextAlignments : Array[TAlignment] of Word = (dt_Left, dt_Right, dt_Center);
Var
  HorAlignment : TAlignment;
  VerAlignment : TVerticalAlignment;
  Texto : string;
  Altura : integer;
  CRect : TRect;
  options : integer;
begin
  Texto:=Cells[ARow,Acol];
  HorAlignment:=FAlignment;
  VerAlignment:=FVerticalAlignment;
  if Assigned(FOnGetCellAlignment) then FOnGetCellAlignment(Self,ARow,ACol,HorAlinea
  if Assigned(FOnGetCellColor) then
    FOnGetCellColor(Self,ARow,ACol,AState,Canvas.Brush,Canvas.Font);
  Canvas.FillRect(ARect);

  Inc(ARect.Left,2);
  Dec(ARect.Right,2);
  CRect:=Arect;
  options:=TextAlignments[HorAlignment] or dt_VCenter;
  if Multilinha then options:=options or dt_WordBreak;
  if not DefaultDrawing then
    inherited DrawCell(ACol,ARow,ARect,AState)
  else
    with ARect,Canvas do
    begin
      Altura:=DrawText(Handle,PChar(Texto),-1,CRect,options or dt_CalcRect);
      if FVerticalAlignment = vaCenter then
      begin
        if Altura < Bottom-Top+1 then
        begin
          Top:=(Bottom+Top-Altura) shr 1;
          Bottom:=Top+Altura;
        end;
```

```
      end
      else if FVerticalAlignment = vaBottom then
        if Altura < Bottom-Top+1 then Top:=Bottom-Altura;
      DrawText(Handle,PChar(Texto),-1,ARect,options)
    end;
end;
```

A primeira coisa que nós faremos é manter o conteúdo da celula para chamar a variável Texto . Logo os valores são determinados por padrão para as variáveis HorAlignment e VerAlignment porque se o usuário não introduziu um alinhamento particular para a celula em questão estes alinhamentos padrões serão aplicados.

Agora vem uma das chaves : a chamada para os eventos. Se o usuário escreveu um gerenciador para o evento de alinhamento, ele o chama. A mesma coisa acontece para o evento de Cor. Deste modo nós já temos o tratamento específico da celula.

O proximo passo é desenhar o fundo da celula por meio do método FillRect para o qual nós passamos o requadro de desenho desta celula (ARect).

Nós faremos uma pausa agora para explicar agora como nós colocaremos o texto.

Em princípio veja, a coisa lógica seria usar o método TextOut do objeto Canvas. Este método precisa como parâmetros as coordenadas (x ,y) onde colocar a string com o texto. Mas para nossos propósitos é ruim, pois teriamos que calcular a posição correta nas coordenadas (x,y). Nós também teríamos que calcular as divisões de palavras necessárias para as celulas multi-linhas, etc. Em resumo, é um rolo!! Mas nós podemos evitar todo este trabalho graças a uma função API do Windows: DrawText
DrawText precisa dos seguintes parâmetros:

- Um Handle para o objeto que nós queremos desenhar (o Canvas do grid).
- Uma tipo de cadeia terminada em nulo com o texto a desenhar (Pchar(Texto))
- Um número que indica o número de caractere a desenhar (-1 para tudo)
- O retângulo no qual o texto vai se enquadrar (TRect)
- Uma série de opções de formatação de texto. Nós usaremos os controles de alinhamento de texto (dt_Left, dt_Center, dt_Right, dt_VCenter), (dt_WordBreak) e o de cálculo de altura (dt_CalcRect)

Há mais opções para o DrawText, se você quiser mais informações leia na ajuda on-line.

Nós voltamos agora ao fluxo do programa. Depois de encher o fundo da celula com o FillRect, nós copiamos na variável CRect o retângulo original (ARect) e eles preparam as opções com que nós chamaremos o DrawText. Aqui surge um pequeno problema: o HorAlignment é do tipo TAlignment (ta_LeftJustify...) e DrawText não entende este tipo, por isso é necessária uma conversão entre este tipo e o do DrawText . Esta conversão é feita através de uma matriz constante denominado TextAlignments. Logo, se a propriedade Multilinha é true, dt_WordBreak é adicionado às opções do DrawText.

O é feito depois é verificar se o usuário trocou o valor da propriedade DefaultDrawing. Se o valor desta propriedade é false indica que o usuário se encarrega de todo o processo, caso contrario o componente se encarrega do desenho.

Se o componente é encarregado de tudo, nós fazemos a primeira chamada ao DrawText para obter a altura exigida do retângulo da celula. Com esta altura, sempre que possível (o multilinha ajusta o texto

inteiro na celula), ele centraliza o texto na celula (ou no topo/rodapé). Uma vez feito isto, nós chamamos novamente o DrawText de forma que ele pega o lugar, e é colocado o texto no canvas do componente. Veja que para isso usamos ARect como retângulo nesta ocasião.

Estas são as grandes caracteristicas com a operação do método DrawCell.

Outros detalhes.

Por ultimo, quero mencionar alguns detalhes pequenos:

- As propriedades de Opções do StringGrid padrão são redefinidas por padrão para true e as opções goRowSizing e goColSizing para multilinhas do nosso componente true por padrão.
- Como sempre, no construtor nós declaramos os valores padrões defeito para as propriedades diferentes que nós definimos (alinhamento, cor, multilinha).
- Depois no codigo fonte é mostrado um exemplo dos eventos de alinhamento, cores e multilinha.

Codigo Fonte do Componente.

```
unit MultiGrid;       { (c) 1997 by Luis Roche }

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, Grids;

type
  TVerticalAlignment = (vaTop, vaCenter, vaBottom);
  TMultiGridAlignmentEvent=procedure(Sender:TObject;ARow,ACol:LongInt;var HorAlignme
  TMultiGridColorEvent=procedure(Sender:TObject;ARow,Acol:LongInt;AState:TGridDrawSt
  TMultiGrid = class(TStringGrid)
  private
    FAlignment : TAlignment;
    FVerticalAlignment : TVerticalAlignment;
    FMultiLinha : Boolean;
    FOnGetCellAlignment : TMultiGridAlignmentEvent;
    FOnGetCellColor : TMultiGridColorEvent;
    procedure SetAlignment(Valor : TAlignment);
    procedure SetVerticalAlignment(Valor : TVerticalAlignment);
    procedure SetMultiLinha(Valor : Boolean);
  protected
    procedure DrawCell(ACol,ARow : LongInt; ARect : TRect; AState : TGridDrawState);
  public
    constructor Create(AOwner : TComponent); override;
  published
    property Alignment : TAlignment read FAlignment write SetAlignment default taLef
    property VerticalAlignment : TVerticalAlignment read FVerticalAlignment write Se
    property MultiLinha : Boolean read FMultiLinha write SetMultiLinha default False
    property OnGetCellAlignment : TMultiGridAlignmentEvent read FOnGetCellAlignment
    property OnGetCellColor : TMultiGridColorEvent read FOnGetCellColor write FOnGet
    property Options default [goFixedVertLine,goFixedHorzLine,goVertLine,goHorzLine,
  end;

procedure Register;

implementation
```

```
constructor TMultiGrid.Create(AOwner : TComponent);
begin
  inherited Create(AOwner);
  FAlignment:=taLeftJustify;
  FVerticalAlignment:=vaCenter;
  {FColor:=clWindowText;}
  FMultiLinha:=False;
  Options:=[goFixedVertLine,goFixedHorzLine,goVertLine,goHorzLine,goRangeSelect,goRo
end;

procedure TMultiGrid.SetAlignment(Valor : TAlignment);
begin
  if valor <> FAlignment then
  begin
    FAlignment:=Valor;
    Invalidate;
  end;
end;

procedure TMultiGrid.SetVerticalAlignment(Valor : TVerticalAlignment);
begin
  if valor <> FVerticalAlignment then
  begin
    FVerticalAlignment:=Valor;
    Invalidate;
  end;
end;

procedure TMultiGrid.SetMultiLinha(Valor : Boolean);
begin
  if valor <> FMultiLinha then
  begin
    FMultiLinha:=Valor;
    Invalidate;
  end;
end;

procedure TMultiGrid.DrawCell(ACol,ARow : LongInt; ARect : TRect; AState : TGridDraw
Const
  TextAlignments : Array[TAlignment] of Word = (dt_Left, dt_Right, dt_Center);
Var
  HorAlignment : TAlignment;
  VerAlignment : TVerticalAlignment;
  Texto : string;
  Altura : integer;
  CRect : TRect;
  Options : integer;
begin
  Texto:=Cells[ARow,Acol];
  HorAlignment:=FAlignment;
  VerAlignment:=FVerticalAlignment;
  if Assigned(FOnGetCellAlignment) then FOnGetCellAlignment(Self,ARow,ACol,HorAlignm
  if Assigned(FOnGetCellColor) then
    FOnGetCellColor(Self,ARow,ACol,AState,Canvas.Brush,Canvas.Font);
  Canvas.FillRect(ARect);
  Inc(ARect.Left,2);
  Dec(ARect.Right,2);
  CRect:=Arect;
  Options:=TextAlignments[HorAlignment] or dt_VCenter;
```

```
  if Multilinha then Options:=Options or dt_WordBreak;
  if not DefaultDrawing then
    inherited DrawCell(ACol,ARow,ARect,AState)
  else
    with ARect,Canvas do
    begin
      Altura:=DrawText(Handle,PChar(Texto),-1,CRect, options or dt_CalcRect);
      if FVerticalAlignment = vaCenter then
      begin
        if Altura < Bottom-Top+1 then
        begin
          Top:=(Bottom+Top-Altura) shr 1;
          Bottom:=Top+Altura;
        end;
      end
      else if FVerticalAlignment = vaBottom then
        if Altura < Bottom-Top+1 then Top:=Bottom-Altura;
      DrawText(Handle,PChar(Texto),-1,ARect,options)
    end;
end;

procedure Register;
begin
  RegisterComponents('Curso', [TMultiGrid]);
end;

end.
```

- # Exemplo de utilização.

Logo um exemplo de utilização do nosso componente novo:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  MultiGrid1.Cells[0,1]:='Janeiro';
  MultiGrid1.Cells[0,2]:='Fevereiro';
  MultiGrid1.Cells[0,3]:='Total Ano';
  MultiGrid1.Cells[0,4]:='Notas';
  MultiGrid1.Cells[1,0]:='Área A';
  MultiGrid1.Cells[2,0]:='Área B';
  MultiGrid1.Cells[3,0]:='Outras Areas';
  MultiGrid1.Cells[4,0]:='TOTAL';
  MultiGrid1.Cells[1,1]:='1.000.000';
  MultiGrid1.Cells[1,2]:='1.250.000';
  MultiGrid1.Cells[1,3]:='9.150.000';
  MultiGrid1.Cells[1,4]:='Incremento sobre ano anterior';
  MultiGrid1.Cells[2,1]:='1.450.000';
  MultiGrid1.Cells[2,2]:='  950.000';
  MultiGrid1.Cells[2,3]:='4.150.000';
  MultiGrid1.Cells[2,4]:='Decremento';
  MultiGrid1.Cells[3,1]:='4.000.000';
  MultiGrid1.Cells[3,2]:='3.250.000';
  MultiGrid1.Cells[3,3]:='17.250.000';
  MultiGrid1.Cells[3,4]:='Incremento sobre ano anterior';
  MultiGrid1.Cells[4,1]:='6.450.000';
  MultiGrid1.Cells[4,2]:='5.450.000';
  MultiGrid1.Cells[4,3]:='30.550.000';
```

```
  MultiGrid1.Cells[4,4]:='';
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MultiGrid1.Color:=clRed;
end;

procedure TForm1.MultiGrid1GetCellAlignment(Sender: TObject; ARow,
  ACol: Longint; var HorAlignment: TAlignment;
  var VerAlignment: TVerticalAlignment);
begin
  if (ACol in [1..3]) and (ARow in [1..4]) then
    HorAlignment:=taRightJustify
  else HorAlignment:=taCenter;
end;

procedure TForm1.MultiGrid1GetCellColor(Sender: TObject; ARow,
  Acol: Longint; AState: TGridDrawState; Abrush: TBrush; AFont: TFont);
begin
  if (ARow=0) then
  begin
    ABrush.Color:=clMaroon;
    AFont.Color:=clWhite;
    AFont.Style:=[fsBold];
  end
  else if (ACol=0) then
  begin
    AFont.Color:=clBlack;
    AFont.Style:=[fsBold];
    ABrush.Color:=clYellow;
  end
  else
  begin
    AFont.Color:=clBlack;
    ABrush.Color:=clYellow;
  end;
end;
```

Luis Roche revueltaroche@redestb.es

Ultima modificación 20.12.1996