Delphi – TUTORIAL
StringGrid

Version 1.1 – 26.12.2001 (= 2. Weihnachtsfeiertag)

Einleitung

Nun, da ich immer wieder Fragen über die StringGrid-Komponente in Foren gefunden habe, will ich sie hier einmal näher beleuchten. Ich gehe mal davon aus, dass ihr euch ein bisschen mit Delphi auskennt und mindestens Borland Delphi 4 habt, in Version 3 müsste man `ARow` durch `Row` und `ACol` durch `Col` ersetzen.

Die Komponente StringGrid

Das StringGrid ist auf der zweiten Reiterseite, „Zusätzliches“, der Komponentenpalette zufinden, dort wird sie durch den Icon mit dem kleinen Gitternetz im Hintergrund und den Buchstaben „ABC“ im Vordergrund symbolisiert. Mit ihr kann man im Prinzip sein eigenes Tabellenkalkulationsprogramm alla „*Excel*“ programmieren. Nun können wir sie auf dem Formular platzieren, und dann gehen wir mal die wichtigsten Eigenschaften, die uns der Objektinspektor anbietet, durch.
Eins muss ich hier noch, für alle die kein Englisch können, sagen: „`Cell`“ bedeutet Zelle, „`Row`“ heißt Reihe oder Zeile und „Column“ Säule oder passender Spalte, diese Begriffe müssten euch aus anderen Programmen bekannt sein.

`ColCount` und `RowCount`	Gibt die Spalten- bzw. Zeilenanzahl an
`DefaultColWidth` und `DefaultRowHeight`	Gibt die Standardwerte für die Weite/Höhe einer Spalte/Zeile an
`DefaultDrawing`	Gibt an, ob die Zellen automatisch gezeichnet werden, ist mit dem Befehl `DrawCell` und dem Ereignis `OnDrawCell` verbunden
`FixedCols`, `FixedRows` und `FixedColor`	Hier wird angegeben, ob ihr fixierte(= unveränderbare) Zeilen und Spalten wollt und ihre Farbe, geeignet für Überschriften und zum Verschieben der Zellenhöhe und Breite
`GridLineWidth`	Gibt die Breite der Line zwischen den Zellen in Pixel an

Wichtig ist auch die Eigenschaft „`Options`“ die aber in der Hilfe gut erklärt ist, ich übernehme es also einfach mal:

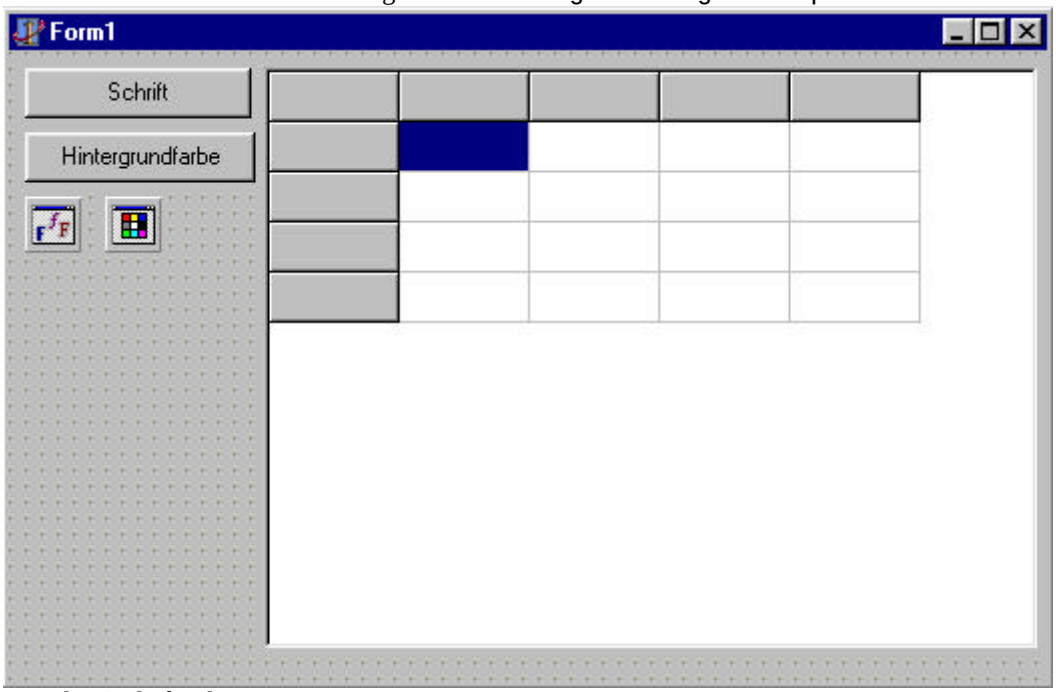
`goFixedVertLine`	Die fixierten Spalten des Gitters werden durch vertikale Linien getrennt.
`goFixedHorzLine`	Die fixierten Zeilen des Gitters werden durch horizontale Linien getrennt.
`goVertLine`	Die verschiebbaren Spalten des Gitters werden durch vertikale Linien getrennt.
`goHorzLine`	Die verschiebbaren Zeilen des Gitters werden durch horizontale Linien getrennt.
`goRangeSelect`	Der Benutzer kann Zellbereiche auswählen. Falls `Options` den Wert `goEditing` enthält, wird `goRangeSelect` ignoriert. Dies gilt nicht, wenn `goRowSelect` in `Options` enthalten ist.
`goDrawFocusSelected`	Die Zellen mit dem Eingabefokus werden ebenso wie die ausgewählten Zellen ohne Eingabefokus in einer bestimmten Farbe angezeigt. Ist der Wert `goDrawFocusSelected` nicht enthalten, wird die Zelle mit dem Eingabefokus durch ein Fokusrechteck und nicht durch eine spezielle Hintergrundfarbe gekennzeichnet. Dies gilt nicht, wenn `goRowSelect` in `Options` enthalten ist.
`goRowSizing`	Die verschiebbaren Zeilen können einzeln vergrößert bzw. verkleinert werden.
`goColSizing`	Die verschiebbaren Spalten können einzeln vergrößert bzw. verkleinert werden.
`goRowMoving`	Die verschiebbaren Zeilen können mit der Maus verschoben werden.
`goColMoving`	Die verschiebbaren Spalten können mit der Maus verschoben werden.
`goEditing`	Der Inhalt der Zellen kann bearbeitet werden. Ist `goEditing` in

	Options enthalten, hat goRangeSelect keine Auswirkung.
goTabs	Der Benutzer kann die Zellen des Gitters mit TAB und UMSCHALT+TAB aufrufen.
goRowSelect	Es werden gesamte Zeilen ausgewählt und keine einzelnen Zellen. Ist goRowSelect in Options enthalten, hat goAlwaysShowEditor keine Auswirkung.
goAlwaysShowEditor	Der Bearbeitungsmodus kann nicht beendet werden. EditorMode muß nicht mit der Eingabetaste oder F2 aktiviert werden. Ist goEditing in Options nicht enthalten, hat goAlwaysShowEditor keine Auswirkung. Ist goRowSelect in Options enthalten, hat goAlwaysShowEditor ebenfalls keine Auswirkung.
goThumbTracking	Das Bild des Gitters wird aktualisiert, sobald die Positionsmarke in der Bildlaufleiste verschoben wird. Ist goThumbTracking nicht in Options enthalten, wird das Bild erst aktualisiert, nachdem die Maustaste an der neuen Position losgelassen wurde.

So nun wären wir damit durch, aber nun zum wirklich wichtigen Teil, der nicht in der Hilfe zu finden ist.

Erstellen eines StringGrids nach Wunsch

Wir haben also ein StringGrid im Form und wollen das für jede Zelle eine individuelle Schrift und Farbe gewählt werden kann. Dafür brauchen wir wie vorhin schon mal angedeutet da OnDrawCell-Ereignis. Also wählen wir DefaultDrawing = True und erzeugen das OnDrawCell-Ereignis. Dann brauchen wir noch zwei Buttons, mit den Namen „ButtonSchrift“ und „ButtonHintergrundfarbe“, wie leicht zu erraten ist, wählt man über den einen die Schrift und über den anderen die Farbe des Zellenhintergrunds. Als letzte brauchen wir nun noch ein FontDialog und einen ColorDialog. Hier also das Bild, was ihr so in etwa vor euch haben müsstet. Damit man was eingeben kann muß goEditing in den Options = true sein.



Nun der Code für die Unit:

```
////////////////////////////////////////////////////////////////
// DefaultDrawing muß True sein                               //
////////////////////////////////////////////////////////////////

{1}
type TEigenschaften = Record // Speichert die Eigenschaften einer Zelle
     Farbe : TColor;         // Hintergrundfarbe
     Schrift : TFont;        // Schrift
     end;

var  EigenschaftenGrid : array [0..5,0..5] of TEigenschaften; // 5x5 Zellen
```

```
{2}
procedure TForm1.FormCreate(Sender: TObject);
var i,j : integer;
begin
     { - Zum Beginn sollen alle Zellen weiß sein - }
     for i:=0 to 5 do
     begin
          for j:=0 to 5 do
          begin
               EigenschaftenGrid[i,j].Farbe:=clWhite;
          end;
     end;
     { - Links oben in der Ecke soll 0,0 stehen - }
     StringGrid1.Cells[0, 0]:='(0;0)';
end;

{3}
procedure TForm1.StringGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
          Rect: TRect; State: TGridDrawState);
var  outRect: TRect;  // Die Ausmaße der Zelle
     Text : String;   // Der Inhalt der Zelle
begin
     Text := StringGrid1.Cells[ACol, ARow]; // Holt sich den Text der Zelle
     outRect:=Rect;                         // Holt sich die Größe der
Zelle
     { - Säubert die Zelle - }
     StringGrid1.Canvas.Brush.Color:=EigenschaftenGrid[ACol,ARow].Farbe; //
Setzt die Hintergrundfarbe
     StringGrid1.Canvas.Fillrect(Rect);       // Füllt das Kästchen weiß
     if EigenschaftenGrid[ACol,ARow].Schrift <> nil then
     StringGrid1.Canvas.Font.Assign(EigenschaftenGrid[ACol,ARow].Schrift);
// Setzt die Schriftfarbe

     { - Text  ausgeben – }
     DrawText(StringGrid1.Canvas.Handle,
         PChar(Text),
         length(Text),
         outRect,
         DT_Left  //=Linksbündig, DT_Center=Zentiert, DT_Right=Rechtsbündig
         or
         DT_WordBreak); //=Umbruch, DT_SingleLine=ohne Umbruch
end;

{4}
procedure TForm1.ButtonSchriftClick(Sender: TObject);
begin
     FontDialog1.Execute;
     EigenschaftenGrid[StringGrid1.Col,StringGrid1.Row].Schrift :=
     FontDialog1.Font;
     StringGrid1.repaint; // Soll gleich aktualisiert werden
end;

procedure TForm1.ButtonHintergrundFarbeClick(Sender: TObject);
begin
    ColorDialog1.Execute;
    EigenschaftenGrid[StringGrid1.Col,StringGrid1.Row].Farbe:=
    ColorDialog1.Color;
    StringGrid1.repaint; // Soll gleich aktualisiert werden
end;
```

Nun besteht bestimmt eine Menge Verwirrung, also will ich es mal erklären.

{1}

Hier erstellen wir uns einen Array in dem wir für jeder der Zellen andere Eigenschaften speichern können.

{2}
Wie das Kommentar schon sagt, wir setzen erst mal die Grundfarbe. Interessant ist auch das Setzen des Textes in der linken, oberen Ecke, mittels zwei Schleifen könnte man so die Spalten und Zeile nummerieren. `StringGrid1.Cells[x, y]` enthält den Text für eine bestimmte Zelle, die Zellen sind von linksoben beginnend bei 0,0 nummeriert.
{3}
Das eigentliche Herzstück, erinnert vielleicht einige an das zeichnen auf dem `Canvas`, ist auch ähnlich. Ich denke man kann es mit Hilfe der Kommentare recht gut verstehen. Das wichtigste ist die Windows-API-Funktion `DrawText`, sie braucht folgende Parameter: den Text, die Länge des Textes, den Raum (Platz für die Ausgabe als Rechteck) und die Formatierung. Für die Formatierung gibt es mehrere Möglichkeiten, die sinnvollen habe ich immer im Kommentar erwähnt, sie können mit `or` verknüpft werden, weitere findet man in der englischsprachigen API-Hilfe unter `DrawText` (z.B.: `DT_VCenter or DT_Center or DT_SingleLine` = Text einzeilig und vertikal und horizontal zentriert). Diese Parameter der Textausgabe werden auch als „Flags“ bezeichnet, Flags können immer mit `or` verknüpft werden.
{4}
Hiermit wird der Auswahldialog für die Schrift oder die Hintergrundfarbe aufgerufen.

Benötigte Zeilenbreite auslesen und einstellen

Bei der Lösung oben kann man das Problem haben, dass Teile des Textes abgehackt werden, dem kann man entgegen wirken indem man die Zelle automatisch verbreitert. Man benutzt die gleiche Prozedur wie oben, liest ab mit Hilfe von `DrawText` und der Flag `DT_CalcRect` die benötige Breite aus und setzt diese. Allerdings muss man für einen reibungslosen Bildaufbau ein paar Kontrollen einbauen, diese speichern die maximale Ausdehnung, um ein Zurücksetzen bei einer kleineren Zelle zu verhindern. Hier also ein modifizierte Version, der neue Code ist hervorgehoben.

```
...
var  EigenschaftenGrid : array [0..5,0..5] of TEigenschaften; // 5x5 Zellen
     maxX, maxY : array [0..5] of integer; // gegen ein Verkürzen
procedure TForm1.FormCreate(Sender: TObject);
var i,n : integer;
begin
     { - Zum Beginn sollen alle Zellen weiß sein - }
     for i:=0 to 5 do
     begin
          // - Setzen auf 1 -
          maxX[i]:=1;
          maxY[i]:=1;
          for n:=0 to 5 do
          begin
               EigenschaftenGrid[i,n].Farbe:=clWhite;
          end;
     end;
     { - Links oben in der Ecke soll 0,0 stehen - }
     StringGrid1.Cells[0, 0]:='(0;0)';
end;

procedure TForm1.StringGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
          Rect: TRect; State: TGridDrawState);
var  outRect: TRect;  // Die Ausmaße der Zelle
     Text : String;   // Der Inhalt der Zelle
     x, y : Integer;  // Die Breite und Höhe
begin
     Text := StringGrid1.Cells[ACol, ARow]; // Holt sich den Text der Zelle
     outRect:=Rect;                         // Holt sich die Größe der
Zelle
     { - Säubert die Zelle - }
     StringGrid1.Canvas.Brush.Color:=EigenschaftenGrid[ACol,ARow].Farbe; //
Setzt die Hintergrundfarbe
     StringGrid1.Canvas.Fillrect(Rect);       // Füllt das Kästchen weiß
     if EigenschaftenGrid[ACol,ARow].Schrift <> nil then
     StringGrid1.Canvas.Font.Assign(EigenschaftenGrid[ACol,ARow].Schrift);
// Setzt die Schriftfarbe
```

```
      // Abfrage
      DrawText(StringGrid1.Canvas.Handle,
          PCHar(Text),
          length(Text),
          outRect,
          DT_CalcRect
          or
          DT_Left  // der gleiche Textsatz der auch beim OnDraw-Ereigniss
          or       // verwendet wird
          DT_WordBreak);
      x := OutRect.Right  - OutRect.Left; // Die Weite
      y := OutRect.Bottom - OutRect.Top;  // Die Höhe
      if maxX[ACol]<X then
      begin
        maxX[ACol]:=X; // Maximale festlegen
        Stringgrid1.ColWidths[ACol]   := x;
        OutRect.Right:=OutRect.Left+X;
      end;
      if maxY[ARow]<Y then
      begin
        maxy[ARow]:=y; // Maximale festlegen
        Stringgrid1.RowHeights[ARow]  := y;
        OutRect.Bottom:=OutRect.Top+Y;
      end;

      { - Text  ausgeben - }
      DrawText(StringGrid1.Canvas.Handle,
          PChar(Text),
          length(Text),
          outRect,
          DT_Left  //=Linksbündig, DT_Center=Zentiert, DT_Right=Rechtsbündig
          or
          DT_WordBreak); //=Umbruch, DT_SingleLine=ohne Umbruch
end;
```

Zeile löschen

Hier gebe ich euch mal eine Übersicht, wie man alltägliche Probleme mit dem StringGrid lösen kann.
Als erstes wollen wir eine Zeile löschen, das läuft recht einfach wir löschen sie und lassen den Rest nachrücken.
Wir nehmen also diese Prozedur.

```
procedure GridDelRow(RowNumber : Integer; Grid : TStringGrid);
var   i : Integer;
begin
  Grid.Row := RowNumber;
  if (Grid.Row = Grid.RowCount -1) then
  begin
    // Falls letzte Zeile
    Grid.RowCount := Grid.RowCount - 1;
  end
  else
  begin
    // Wenn doch nicht die letzte Zeile
    for i := RowNumber To Grid.RowCount - 1 do
    begin
      Grid.Rows[i] := Grid.Rows[i+ 1];
    end;
    Grid.RowCount := Grid.RowCount - 1;
  end;
end;
```

Nun sag ich euch noch schnell die Benutzung: `GridDelRow(3,StringGrid1)`- jetzt würde die dritte Zeile unseres Gitters gelöscht werden. Ich denke, dass es kein Problem ist es für eine Spalten umzustellen (`Row` durch `Col` ersetzen).

Spalte hinzufügen

Jetzt fügen wir mal - zur Abwechselung - eine Spalte hinzu. Also hier wieder der Code:

```
procedure GridAddCol (NewColumn: Integer; Grid: TStringGrid);
var Column: Integer;
begin
  Grid.ColCount := Grid.ColCount+1;
  for Column := Grid.ColCount-1 downto NewColumn do
    Grid.Cols[Column].Assign(Grid.Cols[Column-1]);
  Grid.Cols[NewColumn-1].Text := '';
end;
```

Nun wieder ein Beispielaufruf `GridAddColumn(3, StringGrid1)`, aber ACHTUNG für das obige Beispiel könnt ihr Probleme bekommen, wenn ihr außerhalb des Arrays landet. Das auf die Zeile zu münzen müsste wieder leicht sein – oder?

Zeilen tauschen

Da ist nicht aufregendes bei, es wird nur ein Buffer verwendet.

```
procedure StringGridZeilenVertauschen(StringGrid: TStringGrid; i, j :
Integer);
var
  Temp: TStrings;
begin
  Temp:= TStringList.Create;
  With StringGrid do begin
    Temp.Assign(Rows[i]);
    Rows[i].Assign(Rows[j]);
    Rows[j].Assign(Temp);
  end;
  Temp.Free;
end;
```

Wieder eine Anwendung: `StringGridZeilenVertauschen(Stringgrid1,1,2);`

De-/Selektieren

Was der Benutzer kann, können wir mit unserem Programm schon lange, hier zeige ich euch also wie man was selektiert.

```
StringGrid1.Selection := TGridRect(Rect(1,1,2,4));
```

Aber auch hier gibt es was zu beachten `goRangeSelection` muß true sein. Deselektieren geht genauso einfach:

```
StringGrid1.Selection := TGridRect(Rect(-1,-1,-1,-1));
```

Liniengestalt ändern

Nehmen wir an, wir wollen einen Rechnungsstrich zeichnen, dann brauchen wir unser geliebtes `OnCellDraw`.

```
procedure TForm1.StringGrid1DrawCell(Sender: TObject; ACol,
  aRow: Integer; Rect: TRect; State: TGridDrawState);
begin
  if (aRow = 3) and not (gdFixed In State) then
  begin
    with (sender as TStringGrid).Canvas do
    begin
      Pen.Color := clBlack;
      Pen.Width := 2;
      Pen.Style := psSolid;
      MoveTo( rect.left-1, rect.bottom );
      Lineto( rect.right-1, rect.bottom );
```

```
    end;
  end;
end;
```

Ich klemme mir mal meine Erklärung, ich sage nur 3. Zeile und `Pen` enthält die Eigenschaften für alle Zeichnungen wie z.B.: `LintTo`.

Zusammenfügen von Zellen

Okay, hier mal wieder auch ein Trick mit dem `OnDrawCell`-Ereigniss.

```
procedure TForm1.StringGrid1DrawCell(Sender: TObject; ACol,
  ARow: Integer; Rect: TRect; State: TGridDrawState);
var i, x, y: Integer;
begin
  if gdFixed in State then exit;
  if ARow > 1 then exit;
  with sender as TStringGrid do
  begin
    if aCol < Pred(ColCount) then Rect.Right := Rect.Right + GridlineWidth;
    y:= Rect.Top + 2;
    x:= Rect.Left + 2;
    for i:= 1 to aCol-1 do x:= x - ColWidths[i] - GridlineWidth;
    Canvas.Brush.Color := clWhite;
    Canvas.Brush.Style := bsSolid;
    Canvas.FillRect(Rect);
    Canvas.TextRect(Rect, x, y, Cells[1,1] );
  end;
end;
```

Nun kann man nur in das erste Kästchen eintragen, ansonsten wird die ganze Reihe verschmolzen. Da bietet es sich doch an, wenn man gar nicht erst in die anderen Kästchen schreiben kann und das folgt jetzt.

ReadOnly für einzelne Zellen

Diesmal brauchen wir ein anderes Ereignis, `das OnSelectCell`-Ereigniss.

```
procedure TForm1.StringGrid1SelectCell(Sender: TObject;
   ACol, ARow: Integer; var CanSelect: Boolean);
begin
  if ACol <> 2 then
    Stringgrid1.Options:=Stringgrid1.Options + [goEditing]
  else
    Stringgrid1.Options:= Stringgrid1.Options - [goEditing];
end;
```

Wenn man nun will das die Zelle nicht einmal ausgewählt werden kann, nimmt man:

```
procedure TForm1.StringGrid1SelectCell(Sender: TObject; ACol,
  ARow: Integer; var CanSelect: Boolean);
begin
   if ACol = 2 then CanSelect:=False;
end;
```

Beide male ist die Spalte Nummer 2 (eigentlich ja 3, wir zählen null ja nicht mit) gesperrt.

Scrollen im Gleichtakt

Nun haben wir mal ein neues Ereignis, `OnTopLeftChange`, es wird ausgelöst, sobald in der oberen, linken Ecke ein anders Kästchen auftaucht, vereinfacht: wenn gescrollt wurde. So können wir also zwei Grids synchronisieren, da muss man für sie einfach diesen Code bei der Ereignisbehandlung reinschreiben.

```
procedure TForm1.StringGrid1TopLeftChanged(Sender: TObject);
begin
   StringGrid2.TopRow  := StringGrid1.TopRow;
   StringGrid2.LeftCol := StringGrid1.LeftCol;
end;
```

Will man das es auch reagiert, wenn das andere StringGrid(2) bewegt wird, muss man nur 1 und 2 vertauschen. Es ist auch denkbar, dass ein StringGrid gar keine ScrollBars haben soll und nur mitscrollt.

Ein Bild in das StringGrid zeichnen

Nun zeige ich euch noch wie man ein Bild ausgeben kann, natürlich brauchen wir wieder mal das `OnCellDraw`-Ereignis.

```
procedure TForm1.StringGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
begin
  if ARow=0 then
    StringGrid1.Canvas.Draw(Rect.Left,Rect.Top, Image1.Picture.Bitmap);
end;
```

Dafür brauchen wir jetzt noch ein Image und es empfiehlt sich wieder die Reihe auf ReadOnly zusetzen.

Kippen des Inhaltes

Manchmal kann es nützlich sein, den Inhalt des StringGrids zu kippen, also Spalten und Zeilen umzukehren, man könnt es auch Achsenvertauschung nennen.

```
procedure SwapGrid(grd:TStringgrid);
var tmp:TStringgrid;
      x:integer;
begin
   tmp:=TStringGrid.create(nil);
   tmp.colcount:=grd.rowcount;
   tmp.rowcount:=grd.colcount;
   for x:=0 to grd.colcount-1 do
      tmp.rows[x]:=grd.cols[x];
   grd.colcount:=tmp.colcount;
   grd.rowcount:=tmp.rowcount;
   for x:=0 to grd.colcount-1 do
      grd.cols[x]:=tmp.cols[x];
end;
```

Eine Anwendung: `SwapGrid(Stringgrid3);`, aber Achtung die Zeilenhöhen und -breiten werden nicht getauscht.

Auf Enter in die nächste Zelle

Manchmal ist es ganz nützlich nach Drücken der Entertaste in das nächste Feld zu springen. Hier im Beispiel hüpfen wir nach rechts und wenn das nicht mehr geht in die nächste Zeile. Der Code kommt in die `OnKeyDown`-Behandelung.

```
procedure TForm1.StringGrid1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
if Key=VK_RETURN then                                  // Enter gedrückt
   begin
    if StringGrid1.Col = StringGrid1.ColCount-1 then // Letzte Zelle?
     begin
      if StringGrid1.Row<StringGrid1.RowCount-1 then // Noch eine Reihe
hier drunter?
      begin                                            // Wenn ja dann
         StringGrid1.Col := 0+StringGrid1.FixedCols; // Wieder erste Spalte
(Nicht in den gefixten landen)
         StringGrid1.Row := StringGrid1.Row+1;       // Nächst Zeile
      end;
    end
    else
    StringGrid1.Col:=StringGrid1.Col+1;              // Wir sind nicht am
Zeilenende, also einen nach rechts
   end;
end;
```

Ein Extra-Hint für jede Zelle

Eventuell kann es ganz nützlich sein für jede Zelle einen gesonderten Hint zu haben, es gibt dort allerdings einen Trick mit dem ist es kein größeres Problem mehr und der Hint wird noch vor dem Verlassen des Grids aktualisiert. Dazu benutzen wir die `OnMouseMove`-Behandelung. Natürlich muss `ShowHint` für das StringGrid auf `true` gesetzt werden.

```
var
  LastCol, LastRow : longint;
...
procedure TForm1.StringGrid1MouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
var ACol, ARow: longint;
begin
  StringGrid1.MouseToCell(X, Y, ACol, ARow);
  StringGrid1.Hint:=Sie sind in der Zelle: '
                           +IntToStr(ACol)+'-'+IntToStr(ARow);
  if (ACol<>LastCol) or (ARow<>LastRow) then
  begin
    Application.CancelHint;
    LastCol:=ACol;
    LastRow:=ARow;
  end;
end;
```

`LastCol` und `LastRow` müssen außerhalb des Ereignisses deklariert werden, da durch sie ein ständiges übermalen des Hints verhindert wird.

Speichern und Laden eines StringGrids

Das ist schon etwas schwerer, aber ich habe hier zwei Prozeduren, mit denen das schnell von der Hand geht. Hier die Prozedur zum Speichern, es wird auch die Zeilenhöhe und -breite mitgespeichert.

```
procedure GridSpeichern(grd:TStringGrid;Datei:string);
var sl:TStringlist;
x,y:integer;
begin
   sl:=TStringlist.create;
   sl.add(inttostr(grd.colcount));
   sl.add(inttostr(grd.rowcount));
   for x:=0 to grd.ColCount-1 do
      for y:=0 to grd.RowCount-1 do
         sl.add(grd.cells[x,y]);
   for x:=0 to grd.ColCount-1 do
         sl.add(inttostr(grd.ColWidths[x]));
   for x:=0 to grd.RowCount-1 do
         sl.add(inttostr(grd.RowHeights[x]));
sl.add(inttostr(grd.clientwidth));
sl.add(inttostr(grd.clientheight));
sl.add(inttostr(ord(grd.ScrollBars)));
sl.savetofile(datei);
sl.free;
end;
```

Die Prozedur zum Laden, hat als letztes den Parameter, ob auch die Höhe und Breite wiederhergestellt werden sollen.

```
procedure GridLaden(grd:TStringGrid;Datei:string;angleichen:boolean);
var sl:TStringlist;
    x,y,z:integer;
begin
   sl:=TStringlist.create;
   sl.loadfromfile(datei);
   grd.colcount:=strtoint(sl.strings[0]);
   grd.rowcount:=strtoint(sl.strings[1]);
   z:=2;
   for x:=0 to grd.ColCount-1 do
      for y:=0 to grd.RowCount-1 do
      begin
         grd.cells[x,y]:=sl.strings[z];
         inc(z);
      end;
   if angleichen then
   begin
      for x:=0 to grd.ColCount-1 do
      begin
         grd.ColWidths[x]:=strtoint(sl.strings[z]);
         inc(z);
      end;
      for x:=0 to grd.RowCount-1 do
      begin
         grd.RowHeights[x]:=strtoint(sl.strings[z]);
         inc(z);
      end;
      grd.clientwidth:=strtoint(sl.strings[z]);
      grd.clientheight:=strtoint(sl.strings[z+1]);
      grd.ScrollBars:=TScrollStyle(strtoint(sl.strings[z+2]));
   end;
sl.free;
end;
```

Nun wieder ein Beispiel, erst Speichern: `GridSpeichern(StringGrid1,'save.grd');` ,dann Laden: `GridLaden(StringGrid1,'save.grd',true);`. Eine genaue Erklärung wäre zu lang, aber ich gebe mal den Leitfaden, eine StringListe die am Anfang, die Anzahl der Spalte und Zeilen hat, dann deren Inhalt und am Schluss die Breiten und Höhen der Zellen und dann die gesamte Ausdehnung speichert.

Drucken eines StringGrids

An dieser Stelle steigern wir die Länge und Schwierigkeit noch einmal, diese Druck-Routine ist seit der letzten Tutorialversion stark verändert, leider ist sie immer noch nicht für mehrseitiges Drucken und auch die Farbunterstützung ist noch lange nicht perfekt. Falls jemand diese Prozedur perfektioniert oder eine bessere schreibt/kennt (besonders mit vernünftiger mehrseitigen Druckunterstützung), hätte ich großes Interesse, meine Mail-Adresse steht unten. Eine Erklärung würde den Rahmen diese ohnehin rechtlangen Tutorials einfach sprengen.

```
procedure GridDruck(grd                                : TStringGrid;
                    links, oben,
                    vSpalte, bSpalte, vZeile, bZeile : integer;
                    scal                               : double;
                    farbig                             : boolean);
var
                    x, y ,li ,ob ,re ,un ,waag ,senk ,a : integer;
                    fix, grund, schrift                   : TColor;
                    r                                     : Trect;

function rech(i,j:integer):integer;
begin
     result:=round(((i*j) / 72) * scal);
end;
```

```
begin
     if vZeile < 0  then vZeile :=0;
     if vSpalte < 0 then vSpalte:=0;
     if (bZeile  >= grd.rowcount) or (bZeile < 0)  then bZeile
:=grd.rowcount - 1;
     if (bSpalte >= grd.colcount) or (bSpalte < 0) then
bSpalte:=grd.colcount - 1;
     if vZeile > bZeile then
     begin
          a:=vZeile;vZeile:=bZeile;bZeile:=a;
     end;
     if vSpalte > bSpalte then
     begin
          a:=vSpalte;vSpalte:=bSpalte;bSpalte:=a;
     end;
     if (scal > 0)and(vZeile < grd.rowcount)and(vSpalte < grd.colcount)
then
     begin
          if farbig then
          begin
               fix:=grd.fixedcolor;
               grund:=grd.color;
               schrift:=grd.font.color;
          end else
          begin
               fix:=clsilver;
               grund:=clwhite;
               schrift:=clblack;
          end;
          waag:=getdevicecaps(printer.handle,logpixelsx);
          senk:=getdevicecaps(printer.handle,logpixelsy);
          links:=rech(links,waag);
          oben:=rech(oben, senk);
          li:=getdevicecaps(printer.handle,physicaloffsetx)+1+links;
          a:=rech(3,waag);
          with printer do
          begin
               title:='Grid-Druck';
               BeginDoc;
               if grd.gridlinewidth > 0 then
               begin
                    canvas.pen.color:=$333333;
                    canvas.pen.width:=1;
                    canvas.pen.style:=pssolid
               end else
               canvas.pen.style := psclear;
               canvas.font      := grd.font;
               canvas.font.color:= schrift;
               canvas.font.size := round((grd.font.size / 0.72) * scal);
               for x:=vSpalte to bSpalte do
               begin
                    ob:=getdevicecaps(printer.handle,
                                      physicaloffsety)+1+oben;
                    re:=li+rech(grd.ColWidths[x]+1,waag);
                    for y:=vZeile to bZeile do
                    begin
                         un:=ob+rech(grd.RowHeights[y]+1,senk);
                         if (x < grd.fixedcols)or(y < grd.fixedrows) then
                             canvas.brush.color:=fix
                         else canvas.brush.color:=grund;
                         canvas.rectangle(li,ob,re+2,un+2);
                         r:=rect(li+a,ob+1,re-a,un-2);
```

```
                        drawtext(canvas.handle,pchar(grd.Cells[x,y]),
                                 length(grd.Cells[x,y]),
                                 r,DT_SINGLELINE or DT_VCENTER);
                        ob:=un;
                    end;
                li:=re;
                end;
            enddoc;
            end;
     end;
end;
```

Nun gut, ich muss noch sagen, dass ihr die Printer-Unit benötigt. Also:

```
Uses
  Printers, Grids; // und andere
```

Nun zur Verwendung, die Prozedur hat neun Parameter:
StringGridName, linker Rand, oberer Rand, die Anfangsspalte, letzte zu druckende Spalte, Anfangszeile und Endzeile, Saklierung und ob es ein Farb- oder Schwarz/Weiß-Druck wird. Hier also ein paar Beispiele:

```
// Ein Grid wird an der äußersten Druckkante in Originalgröße ausgegeben
griddruck(stringgrid1,0,0,-1,-1,-1,-1,1,true);
// Die ersten 5 x 5 Zellen (bei 0 beginnende Zählung) werden mit
// Randabstand wie auf der Form ausgegeben
griddruck(stringgrid1,stringgrid1.left,stringgrid1.top,0,4,0,4,1,true);
// Ein Grid wird in halber Größe und schwarz/weiß ausgegeben
griddruck(stringgrid1,0,0,0,0,-1,-1,0.5,false);
```

Was noch fehlt

Es fehlen noch viele Dinge, viel Fragen bleiben offen, aber hoffentlich keine zum StringGrid in Delphi ;-)
Dies ist nun die zweite, überarbeitete Version und die umfasst jetzt schon 12 DIN-A4 Seiten, daher denke ich die häufigsten Fragen nun endlich aus dem Weg geschafft zu haben, also mache ich hier erst mal Schluss, wenn aber Bedarf und Nachfrage besteht kann ich dieses riesige Tutorial gerne erneut fortführen.
Außerdem will ich an dieser Stelle noch mal den Machern vom „*Easy Delphi Helper 2000 EH online*“ der Homepage www.delphi-treff.de danken.
Ich freue mich über jede E-Mail mit Kritik, Lob oder Wünschen für neue Tutorials.
Mailt an SvenZa@gmx.de und schaut mal wieder auf http://www.hardcoders.de.cx rein.
Tschüß.