

Customising TDrawGrid For Text Grids

by Steve Tung

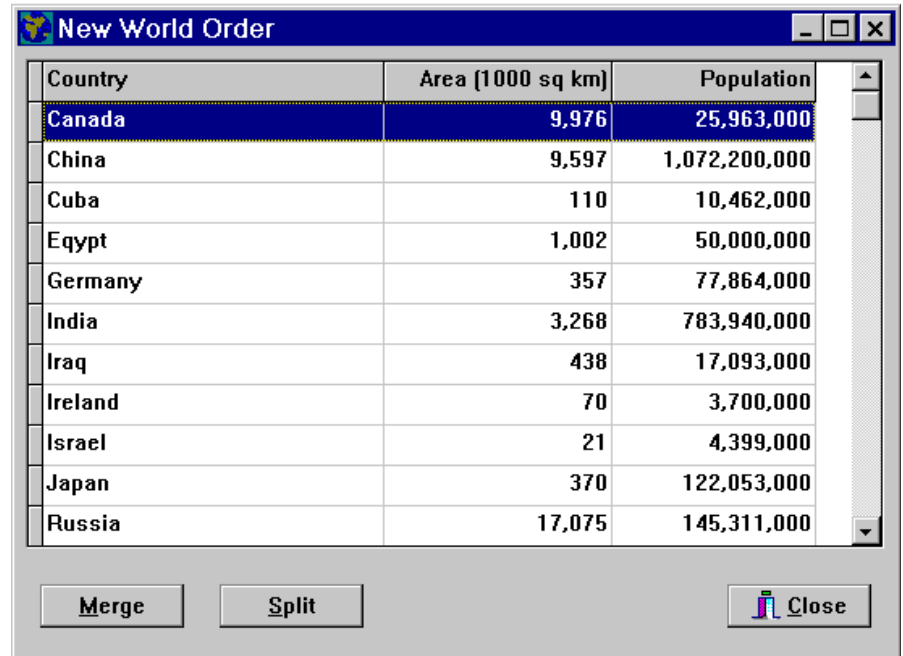
Delphi provides two components for handling tabular data: TStringGrid and TDrawGrid. At first glance, it seems TStringGrid is intended for textual data while TDrawGrid is more suitable for graphical data. This is far from the truth! TDrawGrid is not only capable of handling textual data, it is also more flexible. Unlike TStringGrid, TDrawGrid does not have internal data storage. Instead, users must manage its data externally. The separation of data from the user interface has a great advantage. It gives you the freedom to handle your data with any algorithm.

To demonstrate the use of TDrawGrid, I created a *New World Order* application, which shows a table of a number of countries with some geographical information (Figure 1). There are two functions: Merge combines any number of countries and Split separates a country into two. The user can also sort the table by any column in any direction.

First Attempt

First, let's look at how we would build the application with TStringGrid. Populating the TStringGrid is straightforward enough: just fill the Cells property with the corresponding text. The numeric data is first converted and formatted into strings.

Next we need to sort the table. The user does this by clicking on the heading of any column. Clicking once sorts the column in descending order, clicking the same heading again sorts it in ascending order. To receive mouse clicks on the column headings, I use the OnMouseUp event. OnMouseUp passes the mouse coordinates as parameters. They can be converted into row and column data by using Delphi's built-in MouseToCell procedure. If a heading is clicked,



Country	Area (1000 sq km)	Population
Canada	9,976	25,963,000
China	9,597	1,072,200,000
Cuba	110	10,462,000
Egypt	1,002	50,000,000
Germany	357	77,864,000
India	3,268	783,940,000
Iraq	438	17,093,000
Ireland	70	3,700,000
Israel	21	4,399,000
Japan	370	122,053,000
Russia	17,075	145,311,000

► Figure 1

SortGrid is called to sort the table (see Listing 1).

SortGrid uses a simple bubble sort. Here we begin to feel the inconvenience of TStringGrid. To exchange two rows, we must exchange every cell individually. Worst of all, the algorithm has something wrong. It compares every cell as a string, not as a number. In numeric columns, 1000 is sorted before 500! To convert the formatted string (with comma separators) back into numbers is tedious. It would be much easier if the data was available in the original numeric format.

TDrawGrid

The problem with TStringGrid is that data is stored in the grid's Cells property, which is of type String. Of course this is not the best representation of our data. Instead of putting the data inside TStringGrid, we can control it ourselves and display it using a TDrawGrid. I defined a TCountry class

for each country and a TCountryList class, which descends from TList, as the container for all countries (see Listing 2). You will find an extra field, Selected, in TCountry, which is used later to implement multiple selection.

To display our data with TDrawGrid we use the OnDrawCell event. It is called whenever a grid cell needs to be displayed. In the OnDrawCell event, we only need to determine what text to display and then draw it by calling DrawCellText (see Listing 3). The cell is either a column heading or a data cell. If it is a data cell, its data is available from Countries.Items[Row-1].

DrawCellText looks slightly complicated. It first sets the font color. Then it displays the text by calling the ExtTextOut Windows API function. To add some spice, I included an alignment parameter in DrawCellText. We can now align text on the center or on the right!

Sorting of the different columns is now achieved by sorting the

```

procedure TFrmNewWorld1.GridExchange(
  Grid: TStringGrid; row0, row1: Longint);
begin
  with Grid do
    for i := 1 to ColCount do begin
      s := Cells[i,row0];
      Cells[i,row0] := Cells[i,row1];
      Cells[i,row1] := s;
    end;
  end;
end;

procedure TFrmNewWorld1.SortGrid(ACol: Longint; Desc: Boolean);
var
  i, j: Longint;
begin
  with Grid do
    { bubble sort }
    for i := 1 to RowCount-1-FixedRows do { Remember to minus FixedRows! }
      begin
        for j := RowCount-FixedRows downto i+1 do begin
          if Desc and (Cells[ACol,j] < Cells[ACol,j-1]) then
            GridExchange(Grid,j,j-1);
          if (not Desc) and (Cells[ACol,j] > Cells[ACol,j-1]) then
            GridExchange(Grid,j,j-1);
        end;
      end;
  end;
end;

```

► *Listing 1*

```

TCountry = Class
  Name: String;
  Area, Population: Longint;
  Selected: Boolean;
  constructor Create(AName: String; AArea, APopulation: Longint);
end;

TCountryList = Class(TList)
private
  function Get(index: Integer): TCountry;
  procedure Put(Index: Integer; Item: TCountry);
public
  procedure Sort(ACol: Longint; Desc: Boolean);
  property Items[Index: Integer]: TCountry read Get write Put;
end;

TFrmNewWorld2 = class(TForm)
  ...
  Countries: TCountryList;
end;

```

► *Listing 2*

```

procedure DrawCellText(Sender: TDrawGrid; const Text:
  String; ACol, ARow: Longint; ARect: TRect; State:
  TGridDrawState; Alignment: TAlignment);
begin
  with Sender do begin
    if (ACol = FixedCols) and (ARow = FixedRows) then
      if gdSelected in State then begin
        Canvas.Brush.Color := clHighlight;
        Canvas.Font.Color := clHighlightText;
      end else begin
        Canvas.Brush.Color := Color;
        Canvas.Font.Color := clWindowText;
      end;
    w := Canvas.TextWidth(Text);
    case Alignment of
      taLeftJustify : x := ARect.Left + 2;
      taCenter       : x :=
        (ARect.Left + ARect.Right - w) div 2;
      taRightJustify: x := ARect.Right - w - 2;
    end;
    ExtTextOut(Canvas.Handle,
      x, ARect.Top + 2, ETO_CLIPPED or
      ETO_OPAQUE, @ARect, @(Text[1]), Length(Text), nil);
  end;
end;

```

TCountryList. The TList class does not have a sort method, but it is not difficult to add one. To exchange rows in a TList is much easier than in a TStringGrid as TList provides an Exchange method. Furthermore, we are now correctly comparing the area and population columns as numeric data, not strings! Once the list is sorted, we call the grid's invalidate method to redraw the sorted data.

Before we can proceed to implement the Merge function, I need to solve a big problem. Delphi's grid does not provide multiple selection. The goRangeSelect option allows user to select a range of data, but not separate non-contiguous rows. Although in real life most merging will happen between geographically connected countries, I find it hard to convince users that they are also restricted to merging only adjacent rows! Therefore I must set off to implement multiple selection myself.

Multiple Selection

Although it takes some work, enhancing the grid with multiple selection is not impossible. We must decide how to store the selection, how users can make selections and how to draw them.

Since the grid must allow the selection of any combination of rows, I find it best to embed the

► *Listing 3*

```

procedure TFrmNewWorld2.GridDrawCell(Sender: TObject;
  Col, Row: Longint; Rect: TRect; State: TGridDrawState);
begin
  { Assign text to display }
  Text := '';
  if Row = 0 then { header row }
    case Col of
      1: Text := 'Country';
      2: Text := 'Area (1000 sq km)';
      3: Text := 'Population';
    end
  else { data rows }
    with Countries.Items[Row-1] do begin
      ...
      case Col of
        1: Text := Name;
        2: Text := FormatNum(Area);
        3: Text := FormatNum(Population);
      end;
    end;
  { Set text alignment }
  Alignment := taLeftJustify;
  if Col in [2,3] then
    Alignment := taRightJustify;
  DrawCellText(Sender as TDrawGrid, Text, Col, Row,
    Rect, State, Alignment);
end;

```

selection information in the records themselves. This is what the Selected field in the TCountry class is used for. To test if a record is selected, just check Countries.Items[Row-1].Selected. The original Selection property of the grid is still present, but we will ignore it and use only our selection.

Next, we have to design the user interface. The scheme is simple. Clicking a record selects only that record. Holding the Ctrl key while clicking toggles the selected state of that record. Alternatively, the user may make a selection with the keyboard by pressing the space bar to toggle the selected state of a record. To do this we need to handle OnMouseDown and OnKeyPress events (see Listing 4).

Finally we must draw the selected rows. This is done in the DrawCellText procedure. To show a row as selected, we set Brush.Color and Font.Color to clHighlight and clHighlightText respectively. Note that if the DefaultDrawing property is set (the default), Delphi chooses the color and paints the background for us before calling OnDrawCell. However, it only knows its own Selection! Fortunately DrawCellText paints over the whole cell, so we have the final say in what is really selected.

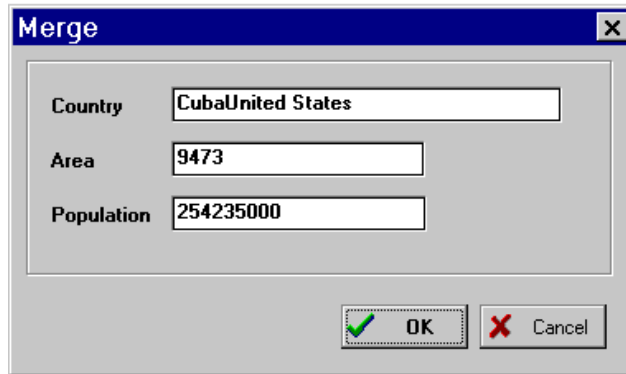
Merge And Split

Finally we come to implement the core functions of the application. Merge totals the area and population of the selected records and suggests a new country name by joining the names of all the participating countries (see Figure 2). Split divides the area and population into two equal parts and suggests two new names. The actual data manipulation is easily done by calling the Add and Delete methods of TList. Again we must invalidate the grid to re-display the data after any change (see Listing 5).

Conclusion

We spent a lot of effort to add features like text alignment and multiple selections to Delphi's grid. But the greatest gain is that we can use TDrawGrid to display our own data. Initially it may need more code

➤ Figure 2



```

procedure TFrmNewWorld2.GridMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  ...
  MouseToCell(X, Y, ACol, ARow);
  ...
  if ssCtrl in Shift then
    with Countries.Items[Grid.Row-1] do
      selected := not selected
    else begin
      for i := 0 to Countries.Count-1 do
        with Countries.Items[i] do
          selected := i = ARow-1;
        end;
      ...
    end;
  procedure TFrmNewWorld2.GridKeyPress(Sender: TObject; var Key: Char);
  begin
    if Key = ' ' then begin
      with Countries.Items[Grid.Row-1] do
        selected := not selected;
      Grid.Invalidate;
    end;
  end;
end;

```

➤ Listing 4

```

procedure TFrmNewWorld2.BtnMergeClick(Sender: TObject);
begin
  ...
  Countries.Add(NewCountry);
  for i := Countries.Count-1 downto 0 do
    if Countries.Items[i].Selected then begin
      Countries.Items[i].Free;
      Countries.Delete(i);
    end;
  ...
  Grid.Invalidate;
end;
procedure TFrmNewWorld2.BtnSplitClick(Sender: TObject);
begin
  ...
  Countries.Items[i].Free;
  Countries.Delete(i);
  Countries.Add(Country1);
  Countries.Add(Country2);
  ...
  Grid.Invalidate;
end;

```

➤ Listing 5

than with TStringGrid, but this will pay off when complicated data manipulation is required. Indeed, data manipulation is so easy that the Merge and Split section is the shortest in this article!

Steve Tung has been working as a Delphi developer in Singapore, but when you read this he will be on an extended break, travelling the world (lucky fellow!).