sgcPivotGrid

Getting Started

Turn flat sales records into a pivot table on a Delphi form, one small demo at a time

About this guide

This guide is for Delphi developers who know how to build a VCL form and want to add a pivot table to it with sgcPivotGrid. You do not need to know anything about pivot tables in advance. If you have ever used a PivotTable in a spreadsheet, you will recognise the ideas at once; if you have not, chapter 1 explains them in a minute.

The guide follows the five QuickStart demos. Each demo adds one idea to the previous one, from a pivot grid with three fields to a small sales analysis application with a field list, filters, drill down and export. Chapter 4 walks through them one at a time: what you will build, the components on the form, the code, how it works and a few changes to try.

Read it with Delphi open next to it. Open a demo, run it, click around in the grid, then read its section and try the suggested changes. Chapters 1 to 3 explain what sgcPivotGrid is, how to install it and the ideas behind it, chapter 5 lists the main components, chapter 6 collects common problems and their fixes, and chapter 7 shows where to go next.

The demos are in the `Demos\PivotGrid\Delphi\00.QuickStart` folder of your sgcUI installation. Each demo has its own subfolder, from `01.HelloPivot` to `05.PivotSalesApp`, with a Delphi project (`.dpr`) that you can open and run directly. All of them are VCL applications for Windows.

Contents

1. What sgcPivotGrid is

The idea in one minute
Your first pivot needs one event handler
What you need
Part of sgcUI
Coming from spreadsheets and data grids

2. Install and the palette

Installing
The packages
The palette page
Help at design time
Editions
Where the demos are
Running your first demo

3. Core concepts

Flat records in, a summary out
The engine lives in Grid.Pivot
Fields and the four areas
Feeding data from code
Feeding data from a dataset
When the pivot recalculates
Totals and the expanded state
Reading the result
Look, themes and format rules
Summary: who does what

4. The QuickStart demos

4.1 Demo 01: Hello Pivot
4.2 Demo 02: Pivot Layout
4.3 Demo 03: Pivot Totals
4.4 Demo 04: Pivot DataSet
4.5 Demo 05: Pivot Sales App

5. Main components

Reference table
The pivot grid
The field list
The data sources
The engine and its fields
Beyond the basics

6. Common problems

Data and layout
Expand, totals and formatting
Editions and compiling

7. Where next

The other Delphi demos
Help topics
On the website
Getting help

1. What sgcPivotGrid is

sgcPivotGrid adds pivot tables to Delphi applications, for the VCL and for FireMonkey. A pivot table takes a long list of flat records, such as one row per sale, and summarises it in a grid: one row per region, one column per year, and in every cell the total of the sales that belong to that region and that year. The user can expand and collapse the levels, filter, sort and drill down from a total to the records behind it.

You drop a `TsgcVCLPivotGrid` on a form, describe the layout in the Object Inspector (which column goes on the rows, which on the columns, which one is summed) and hand it the records, either from your own code or from any `TDataSet`. The grid calculates the subtotals and the grand totals, draws the result and keeps it up to date when the layout or the data changes.

This guide is for Delphi developers who are new to sgcPivotGrid. By the end you will know how the pieces fit together, which properties to reach for first, and how to avoid the mistakes that most people make in their first hour.

The idea in one minute

Imagine thirteen sales records, each with a region, a product and an amount:

Region	Product	Sales
North	Laptop	1200
North	Monitor	320
North	Laptop	1150
South	Laptop	1300
... nine more rows		

A pivot table answers the question "how much did every region sell of every product?" by putting the regions on the rows, the products on the columns and the sum of the sales in the cells. The two North laptop sales, 1200 and 1150, become one cell with 2350. A Grand Total row and a Grand Total column are added for free.

Every pivot is described by the same three choices:

- **Which fields go on the rows** and which go on the columns. Their distinct values, called members, become the row and column headers.
- **Which fields are the values**, and how they are aggregated: a sum, a count, an average, a minimum, a maximum and a few more.
- **Which fields filter** the records before anything is counted, for example only the Online sales.

In sgcPivotGrid each of these choices is a *field*: an item of the `Pivot.Fields` collection with a `FieldName` (the source column) and an `Area` (rows, columns, values or filters). Change the area of a field and the whole table is rebuilt. That is the entire idea; everything else is detail.

Your first pivot needs one event handler

The first QuickStart demo is the smallest pivot there is. The layout is set at design time: three items in `Pivot.Fields`, stored in the form file like this:

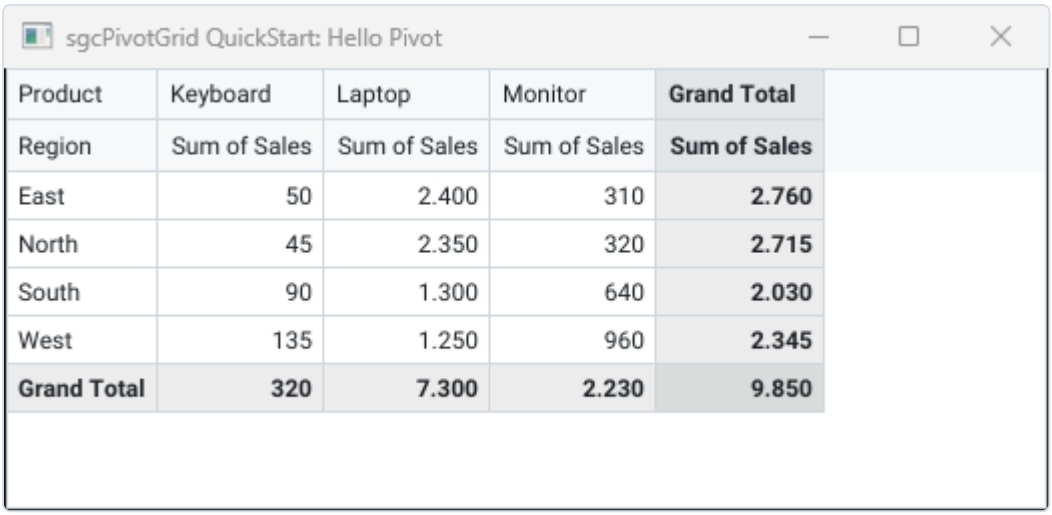
```
  object Grid: TsgcVCLPivotGrid
    Align = alClient
    Pivot.Fields = <
      item
        FieldName = 'Region'
        AreaIndex = 0
      end
      item
        FieldName = 'Product'
        Area = pvaColumn
        AreaIndex = 0
      end
      item
        FieldName = 'Sales'
        Area = pvaData
        AreaIndex = 0
        DisplayFormat = '#,##0'
      end>
  end
```

The data is loaded in `FormCreate`. First the columns of the source records are declared with a name and a type, then the records are appended, all inside a `BeginLoad` and `EndLoad` bracket:

```
procedure TfrmHelloPivot.FormCreate(Sender: TObject);
begin
  Grid.Pivot.BeginLoad;
  try
    Grid.Pivot.DefineColumn('Region', pvtText);
    Grid.Pivot.DefineColumn('Product', pvtText);
    Grid.Pivot.DefineColumn('Sales', pvtFloat);

    Grid.Pivot.AppendRow(['North', 'Laptop', 1200]);
    Grid.Pivot.AppendRow(['North', 'Monitor', 320]);
    // ...
    Grid.Pivot.AppendRow(['West', 'Keyboard', 135]);
  finally
    Grid.Pivot.EndLoad;
  end;
end;
```

That is all. When `EndLoad` runs, the grid matches the field names of the layout with the column names of the data, calculates the table once and paints it. You did not write a single loop over the records, no dictionary of totals and no drawing code.



Product	Keyboard	Laptop	Monitor	Grand Total
Region	Sum of Sales	Sum of Sales	Sum of Sales	Sum of Sales
East	50	2.400	310	2.760
North	45	2.350	320	2.715
South	90	1.300	640	2.030
West	135	1.250	960	2.345
Grand Total	320	7.300	2.230	9.850

Figure 1.1. The result of demo 01: four regions, three products, the sum of the sales in every cell and the grand totals.

What you need

- **Delphi.** The packages in the `Packages` folder cover Delphi 7, 2007, 2009, 2010, XE to XE8, 10 Seattle to 10.4 Sydney, 11, 12 and 13.
- **VCL.** `TsgcVCLPivotGrid` and `TsgcVCLPivotFieldList` run on Windows, Win32 in every version and Win64 where your Delphi version can build it.
- **FireMonkey.** `TsgcFMXPivotGrid` and `TsgcFMXPivotFieldList` have their own packages from Delphi XE6 on, for the platforms FireMonkey targets in your Delphi version.
- **No window at all.** The calculation engine, `TsgcPivotEngine`, uses no visual framework, so a pivot can also run in a console application or a server and be written to PNG, SVG, HTML, CSV or JSON.
- **sgcUICore.** The shared foundation of the sgcUI visual products. It is installed together with sgcPivotGrid, you do not install it separately.

The QuickStart demos are VCL applications. Demo 04 uses a `TClientDataSet` with `MidasLib`, which ships with Delphi, so no extra DLL is needed.

Part of sgcUI

sgcPivotGrid is one product of sgcUI, the eSeGeCe suite of visual components (grids, charts, dashboards, planners and more). All of them draw through sgcUICore, the shared layer with the canvas, the fonts and the light and dark themes. In practice this means two things for you:

- A pivot grid looks the same on the VCL, on FireMonkey and in the headless output, and it matches the other sgcUI controls on your form. The `Theme` property (`utDefault`, `utLight`, `utDark`, `utCustom`, `utHighContrast`) works the same way on all of them.
- The products connect to each other. `TsgcPivotSource_Table` reads the rows of any sgcUI table, such as an sgcGrid, into a pivot, and the result of a pivot can feed a chart. Chapter 7 points to the demos that do this.

Coming from spreadsheets and data grids

If your users export data to a spreadsheet today just to build a PivotTable there, sgcPivotGrid brings that experience into your application: rows, columns, values and filters, subtotals, date grouping, expand and collapse and a field list to drag fields around. The data never leaves your program, and you decide which fields the user may see.

If you know `TDBGrid` or `TStringGrid`, the main difference is that a pivot grid does not show your records; it shows a summary of them. You never fill its cells. You give it records and a layout, and it computes the cells. That is why most of this guide is about the layout (`Pivot.Fields`) and about how the records get in.

2. Install and the palette

This chapter gets sgcPivotGrid onto your Tool Palette and the first demo running. If the components are already installed, skim the palette section and jump to Running your first demo.

Installing

sgcPivotGrid is installed by the sgcUI installer, the same installer for every sgcUI product:

1. Run the sgcUI installer and sign in with the user name and password of your eSeGeCe account.
2. Choose the Delphi versions to install into.
3. On the components page, select **sgcPivotGrid**. The installer shows one checkbox per product and enables the products your licence includes. sgcUICore has no checkbox: it is installed automatically with sgcPivotGrid.
4. Optionally select **sgcPivotGrid for sgcWebSockets**, the bridge used by the AI query component. It installs only when sgcWebSockets is present, and you do not need it for this guide.
5. The installer copies the files, compiles the packages on your machine for each selected Delphi version and installs the design time packages.

Everything goes into one folder, by default `Documents\esegece\sgcUI`. The installer adds its `Source` folder and the folder of the compiled units to the library path of each selected Delphi version, so a new project finds the sgcPivotGrid units without any setting on your side.

The packages

You only need this section if you build the packages yourself. sgcPivotGrid is split into a framework neutral package and one package per framework, so a FireMonkey application never links the VCL and the other way round. Every package name ends with the suffix of the Delphi version, for example `D13` for Delphi 13 or `D7` for Delphi 7.

Package	Kind	Contains
`sgcPivotGrid<V>`	Runtime	The engine, the fields, the sources, the renderer and the export. No VCL, no FireMonkey.
`sgcPivotGridVCL<V>`	Runtime	`TsgcVCLPivotGrid` and `TsgcVCLPivotFieldList`.
`sgcPivotGridFMX<V>`	Runtime, XE6 and later	`TsgcFMXPivotGrid` and `TsgcFMXPivotFieldList`.
`dclsgcPivotGrid<V>`	Design time	Registers the VCL controls and the non visual components on the palette, with the component editor and the FieldName drop down list.
`dclsgcPivotGridFMX<V>`	Design time, XE6 and later	Registers the FireMonkey controls. Install `dclsgcPivotGrid` first.
`sgcPivotGridWS<V>`, `dclsgcPivotGridWS<V>`	Runtime and design time	The sgcWebSockets bridge with `TsgcPivotGridAIQuery`.

Build them in dependency order: sgcUICore first, then `sgcPivotGrid`, then `sgcPivotGridVCL`, then `dclsgcPivotGrid`. Then install the design time package and add `sgcUI\Source` to the library path of your Delphi version (**Tools** > **Options** > **Language** > **Delphi Options** > **Library**).

The palette page

After the installation the Tool Palette has a page named **SGC PivotGrid**. In a VCL project it shows these components:

Component	Visual	What it does
`TsgcVCLPivotGrid`	yes	The pivot grid. It owns a calculation engine, published as `Pivot`.
`TsgcVCLPivotFieldList`	yes	The field list: the user drags fields into the rows, the columns, the values and the filters of a pivot grid.
`TsgcPivotSource_DataSet`	no	Loads a `TDataSet` into a pivot grid and reloads it when the data changes.
`TsgcPivotSource_Table`	no	Loads any sgcUI table, such as the rows of an sgcGrid, into a pivot grid.
`TsgcPivotGridAI`	no	Builds a pivot layout from a question typed in plain language, through an AI provider.
`TsgcPivotChartTable`	no	Exposes the pivot result as a chart table for sgcCharts. Registered editions only.

In a FireMonkey project the same page shows `TsgcFMXPivotGrid` and `TsgcFMXPivotFieldList` instead of the VCL controls, together with the same non visual components. With the sgcWebSockets bridge installed, `TsgcPivotGridAIQuery` is added to the page as well. The QuickStart demos use only the first three components of the table.

Help at design time

Two small design time helpers save you typing:

- **The FieldName drop down list.** In the collection editor of `Pivot.Fields`, the `FieldName` property of a field offers a drop down list of the source columns. When the grid holds no data yet, it lists the columns of a `TsgcPivotSource_DataSet` or `TsgcPivotSource_Table` bound to the grid on the same form. With data loaded only at run time from code, as in demos 01, 02, 03 and 05, the list is empty and you type the names yourself.
- **The component editor.** Right click a pivot grid on the form to find two commands: **Recalculate** recalculates the pivot in the designer, and **Clear Layout** removes every field from `Pivot.Fields`. Use the second one with care, it is meant for starting over.

To edit the layout, select the grid, expand `Pivot` in the Object Inspector and click the ellipsis button of `Fields`. The collection editor that opens lists one item per field; select an item to set its `FieldName`, `Area`, `AreaIndex`, `Caption` and the rest in the Object Inspector.

Editions

sgcPivotGrid is built from the same source for every edition. The Registered and Trial editions include everything. The free Community edition leaves out a few advanced features: calculated fields, show values as, Top N, saving and loading the layout, and the XLSX export, among others. Everything this guide uses is in every edition: the grids, the field lists, the sources, every aggregation, date grouping, totals, filters, sorting, drill down, format rules and the CSV, HTML, JSON, PNG and SVG export.

> **Note.** The QuickStart demos deliberately avoid the advanced features, so they compile and run with every edition. Chapter 7 lists the `AdvancedPivot` demo, which shows the advanced features.

Where the demos are

The demos live in `Demos\PivotGrid\Delphi` under your sgcUI folder. The QuickStart is the first subfolder:

Folder	Project	What it shows
01.HelloPivot	`HelloPivot.dpr`	The smallest pivot: three fields, thirteen records loaded in code
02.PivotLayout	`PivotLayout.dpr`	Two row levels, years from a date column, two values, a layout change in code
03.PivotTotals	`PivotTotals.dpr`	Expand and collapse, grand totals and subtotals, a colour scale and a cell click
04.PivotDataSet	`PivotDataSet.dpr`	A pivot fed by a `TClientDataSet` that follows every posted edit
05.PivotSalesApp	`PivotSalesApp.dpr`	A small sales analysis application: field list, filter, drill down and export

Every demo folder holds the project, its unit and form, and a `readme.html` with a short description. The `readme.html` in `00.QuickStart` itself is the index of the five demos. The other folders in `Demos\PivotGrid\Delphi` are larger demos, listed in chapter 7.

Running your first demo

1. In Delphi, choose **File** > **Open Project** and open `Demos\PivotGrid\Delphi\00.QuickStart\01.HelloPivot\HelloPivot.dpr`.
2. Press **F9**. The window opens with the pivot already calculated.
3. Back in the IDE, open the form, select `Grid` and look at `Pivot.Fields` in the Object Inspector. Those three items are the whole layout.

> **Tip.** To see every property of a form at once, right click the form in the designer and choose **View as Text**. The DFM excerpts in this guide come from that view. Choose **View as Form** to go back to the designer.

3. Core concepts

Nine ideas explain almost everything a pivot grid does. Read them once now, and come back to them when a demo surprises you.

Flat records in, a summary out

A pivot always starts from flat records: a list of rows that all have the same columns, like the result of a SQL query or the lines of a CSV file. One record per sale, per order line, per log entry. You never give the pivot grid totals; you give it the detail and let it total.

Inside, the records are kept in a typed column store. Each source column has a name and a type: `pvtText`, `pvtInteger`, `pvtFloat`, `pvtDateTime` or `pvtBoolean`. The type matters. A date column can be grouped by year or quarter, a number column can be summed, and a text column can only be counted. The type comes from `DefineColumn` when you load from code, or from the field type when you load from a dataset.

The engine lives in Grid.Pivot

The pivot grid is a view. The work is done by a calculation engine, a `TsgcPivotEngine` that the grid creates and publishes as its `Pivot` property. Almost everything you do in code goes through it:

- `Grid.Pivot.BeginLoad`, `DefineColumn`, `AppendRow` and `EndLoad` load the records.
- `Grid.Pivot.Fields` is the layout.
- `Grid.Pivot.RowGrandTotals`, `ColumnGrandTotals`, `SubtotalPosition` and `GrandTotalCaption` control the totals.
- `Grid.Pivot.ExpandAll` and `CollapseAll` open and close the levels.
- `Grid.Pivot.Result` is the calculated table, which you read under a lock.

The grid itself owns only what is about drawing: `Theme`, `RowLayout`, `AlternateRows`, `FormatRules`, the row and header heights, the export methods (`SaveToCSV`, `SaveToHTML`, `SaveToPNG` and more) and the mouse events. When you look for a property and cannot find it on the grid, look under `Pivot`.

Fields and the four areas

The layout is the `Pivot.Fields` collection. You usually edit it in the Object Inspector, and it is saved in the form file. Every item is a `TsgcPivotField`, and these are the properties you will set most often:

Property	Meaning
`FieldName`	The source column. Names are compared ignoring case.
`Area`	`pvaRow` (the default), `pvaColumn`, `pvaData` for a value, or `pvaFilter`.
`AreaIndex`	The position inside the area: 0 is the outer level, 1 the level nested inside it, and so on.
`Caption`	The header text. Empty for a value field means an automatic caption such as "Sum of Sales".
`Aggregation`	For a value field: `pvfSum` (the default), `pvfCount`, `pvfAverage`, `pvfMin`, `pvfMax`, `pvfCountDistinct`, `pvfStdDev`, `pvfVar`.
`DateGrouping`	For a date column: `pvdYear`, `pvdQuarter`, `pvdMonth`, `pvdWeek`, `pvdDay`, or `pvdNone`.
`DisplayFormat`	A `FormatFloat` picture for values, such as `'#,##0'`, or a `FormatDateTime` picture for dates.
`SortBy`, `SortOrder`	Sort the members by their label (`pvsLabel`) or by their value (`pvsValue`), ascending or descending.
`Subtotals`, `Expanded`	Whether the members of this field show a subtotal, and whether they start open. See Totals and the expanded state.
`FilterValues`, `FilterExclude`	The member labels to keep, or to leave out with `FilterExclude`. Empty keeps every record.

Two facts make the layout flexible. First, the same source column can be used by several fields: demo 02 sums *and* averages the Sales column, and demos 03 and 05 put the Date column on the columns twice, once grouped by year and once by quarter. Second, a field whose `FieldName` matches no loaded column is not an error. A row, column or filter field with an unknown name is ignored, and a value field stays empty. This is what lets you design the whole layout at design time, before any data exists.

You can also build the layout in code. Wrap the changes in `BeginUpdate` and `EndUpdate`, so the pivot recalculates once at the end instead of after every property:

```
procedure BuildLayout(Grid: TsgcVCLPivotGrid);
var
  vField: TsgcPivotField;
begin
  Grid.Pivot.BeginUpdate;
  try
    Grid.Pivot.Fields.Clear;

    vField := Grid.Pivot.Fields.Add;
    vField.FieldName := 'Region';
    vField.Area := pvaRow;
    vField.AreaIndex := 0;

    vField := Grid.Pivot.Fields.Add;
    vField.FieldName := 'Date';
    vField.Caption := 'Year';
    vField.Area := pvaColumn;
    vField.AreaIndex := 0;
    vField.DateGrouping := pvdYear;

    vField := Grid.Pivot.Fields.Add;
    vField.FieldName := 'Sales';
    vField.Area := pvaData;
    vField.AreaIndex := 0;
    vField.Aggregation := pvfSum;
    vField.DisplayFormat := '#,##0';
```

```
  finally
    Grid.Pivot.EndUpdate;
  end;
end;
```

The types come from the units `sgcPivotGrid_Types` (the enumerations), `sgcPivotGrid_Fields` (`TsgcPivotField`) and `sgcPivotGrid_VCL` (the grid). To find a field that already exists, `Grid.Pivot.Fields.FieldAt(pvaData, 1)` returns the field at a given area and position, or `nil`.

Feeding data from code

When the records come from your own code, a file or a web service, load them straight into the engine:

```
procedure ReloadData(Grid: TsgcVCLPivotGrid);
begin
  // BeginLoad clears the previous rows (aClear defaults to True)
  Grid.Pivot.BeginLoad;
  try
    Grid.Pivot.DefineColumn('Region', pvtText);
    Grid.Pivot.DefineColumn('Date', pvtDateTime);
    Grid.Pivot.DefineColumn('Sales', pvtFloat);
    Grid.Pivot.AppendRow(['North', EncodeDate(2025, 3, 14), 1200]);
    Grid.Pivot.AppendRow(['South', EncodeDate(2025, 7, 2), 640]);
  finally
    Grid.Pivot.EndLoad;
  end;
end;
```

- `DefineColumn` declares a column with a name and a type. A third parameter sets the caption shown in the field list and in the headers: `DefineColumn('Date', pvtDateTime, 'Order date')` in demo 05.
- `AppendRow` takes the values of one record in the order of the columns. Pass real `TDateTime` and numeric values, not strings.
- `BeginLoad` starts by clearing the rows already loaded, so calling the same procedure again replaces the data.
- The pivot is calculated once, at `EndLoad`. Without the bracket every `AppendRow` would recalculate, which is slow with thousands of records.

Feeding data from a dataset

When the records are in a `TDataSet` (FireDAC, ADO, IBX, dbExpress, a `TClientDataSet` or any other), drop a `TsgcPivotSource_DataSet` on the form and set three properties:

- `DataSet`: the dataset to read.
- `PivotGrid`: the pivot grid to fill.
- `Active = True`: the source loads the dataset as soon as it is open.

The dataset field names become the column names that `FieldName` refers to, and the field types become the column types. With `LiveUpdates`, which is `True` by default, the source reloads the pivot when the dataset reports a data change: a post, an append, a delete, a refresh or a filter change. An edit counts only once it is posted. `Columns` limits the load to some fields (empty means every field), and `Refresh` reloads on demand. Demo 04 shows all of it.

When the pivot recalculates

You rarely call anything to refresh a pivot. With `Pivot.AutoRecalculate`, `True` by default, the engine recalculates by itself:

- at `EndLoad`, after a load bracket;
- after any change to `Pivot.Fields` or to a field property, such as `Area`, `Aggregation` or `FilterValues`, or once at `EndUpdate` when the changes are inside an update bracket;
- after a change to an engine property such as `RowGrandTotals` or `SubtotalPosition`.

Expanding or collapsing a member does not aggregate the data again; it only rebuilds the visible rows and columns, so it is instant even with many records. If you set `AutoRecalculate` to `False`, call `Pivot.Recalculate` yourself.

A recalculation runs on the thread that triggers it, normally the main thread, so in a VCL application you simply change properties in your event handlers.

Totals and the expanded state

When a field has another field nested inside it, its members get a subtotal: in demo 02 every region shows the total of its countries. `Subtotals` on the outer field turns that off, and `Pivot.SubtotalPosition` puts the subtotals above (`pvpTop`, the default) or below (`pvpBottom`) their children. `Pivot.RowGrandTotals` and `Pivot.ColumnGrandTotals` switch the grand totals. Totals are always calculated from the source records, so the total of an average is the average of all the records, never an average of averages.

The `Expanded` property of a field is the starting state of **that field's own members**: whether they show their children. This is the one property beginners get backwards. To start with the quarters hidden under closed years, you set `Expanded = False` on the **Year** field, the outer one, not on the Quarter field. Setting it on the innermost field has no visible effect, because its members have no children to hide.

```
procedure TuneLayout(Grid: TsgcVCLPivotGrid);
var
  vField: TsgcPivotField;
begin
  // show only the Online sales of a filter field
  vField := Grid.Pivot.Fields.FieldAt(pvaFilter, 0);
  if vField <> nil then
    vField.FilterValues.Text := 'Online';

  // start with the years closed: Expanded goes on the Year field
  vField := Grid.Pivot.Fields.FieldAt(pvaColumn, 0);
  if vField <> nil then
    vField.Expanded := False;

  Grid.Pivot.GrandTotalCaption := 'Total';
  Grid.Pivot.EmptyText := '-';
end;
```

At run time the user opens and closes members with the plus and minus buttons, a double click on a row header, or the + and − keys. In code, `Pivot.ExpandAll(pvxRows)` and `Pivot.CollapseAll(pvxColumns)` work on a whole axis. The snippet above also shows `EmptyText`, the text of a cell with no records.

Reading the result

The calculated table is published as a snapshot, `Pivot.Result`, a `TsgcPivotResult`. A recalculation replaces it with a new one and frees the old one, so read it between `Pivot.Lock` and `Pivot.Unlock`:

- The rows and the columns of the table are *items* of an axis: `Result.Item(pvxRows, i)`. An item is a member at some level, a subtotal or the grand total (`Kind = pvkGrandTotal`).
- `Result.NodePath(axis, item.Node)` gives the labels of an item joined by " / ", for example "Europe / Spain".
- `Result.CellText(rowItem, dataCol)` gives the formatted text of a cell.
- With several value fields, every column item has one data column per value: data column = column item × `MeasureCount` + value index.

Demo 03 reads the result in its cell click handler, and demo 05 asks the engine for the source records behind a cell with `GetSourceRows`.

Look, themes and format rules

The look is set on the grid. `Theme` picks one of the sgcUICore themes, shared with every other sgcUI control: `utDefault`, `utLight`, `utDark`, `utCustom` or `utHighContrast`. `AccentColor`, `FontSize`, `RowHeight`, `GridLines`, `AlternateRows` and `TotalsBold` fine tune it. `RowLayout` chooses between the compact layout (`pvlCompact`, all row levels indented in one column) and the tabular layout (`pvlTabular`, one column per row field).

`FormatRules` colours cells by their value without code. A rule with `Kind = pvrValue` applies `BackColor`, `ForeColor` or `Bold` to the values that satisfy its `Operator` and `Value`. A rule with `Kind = pvrColorScale` shades each cell between `MinColor` and `MaxColor`. `Measure` chooses the value field a rule applies to. Colours are `TsgcUIColor` values, 32 bit ARGB numbers such as `$FF63BE7B`, where the first byte is the opacity; 0 means transparent.

Summary: who does what

Piece	Its job
Your code, or `TsgcPivotSource_DataSet`	Puts the flat records into the engine.
`Grid.Pivot` (`TsgcPivotEngine`)	Stores the records, holds the layout and the totals settings, calculates and publishes the result.
`Grid.Pivot.Fields`	The layout: which column goes to which area, how it is grouped, aggregated, sorted and filtered.
`TsgcVCLPivotGrid`	Draws the result with the theme and the format rules, handles the mouse and the keyboard, exports the view.
`TsgcVCLPivotFieldList`	Lets the user change the layout by drag and drop.

4. The QuickStart demos

The five QuickStart demos build on each other. Run them in order: each one adds one or two ideas to the previous one, and the sections below assume you have seen the earlier ones.

Demo	Project	New ideas
01.HelloPivot	`HelloPivot.dpr`	The engine, a layout set at design time, records loaded in code
02.PivotLayout	`PivotLayout.dpr`	Nested row fields, date grouping, two values, changing a field in code
03.PivotTotals	`PivotTotals.dpr`	Expand and collapse, grand totals and subtotals, compact and tabular layout, a colour scale, reading the result
04.PivotDataSet	`PivotDataSet.dpr`	A `TDataSet` as the source, live updates
05.PivotSalesApp	`PivotSalesApp.dpr`	The field list, a filter field, drill down, CSV and HTML export

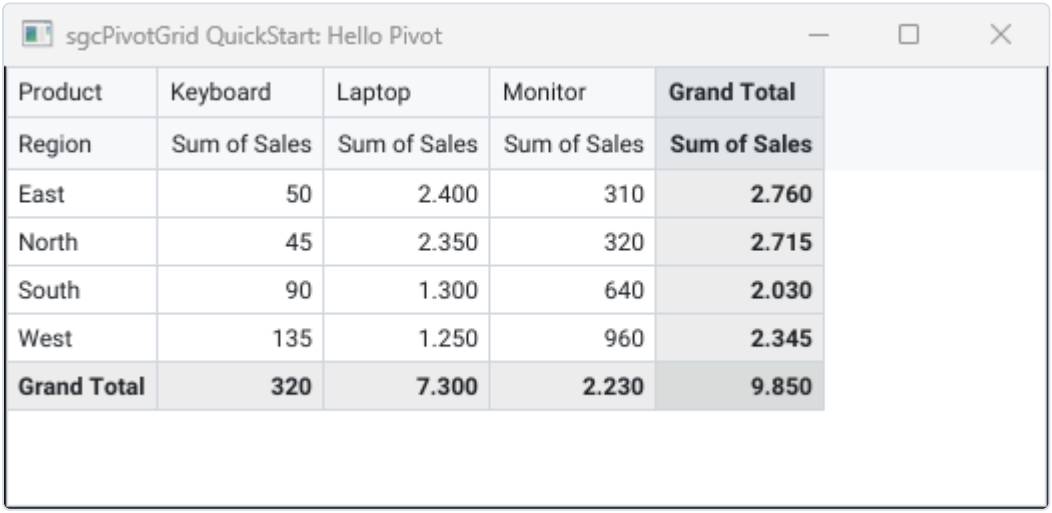
> **Note.** The demos need sgcPivotGrid installed, so the components appear on the SGC PivotGrid palette page and the `sgcUI\Source` folder is on the library path. They use only features that are part of every edition. The numbers in the screenshots use the thousands separator of the machine they were taken on, so on your machine `#,##0` may show commas where the figures show dots.

4.1 Demo 01: Hello Pivot

Folder `01.HelloPivot`, project `HelloPivot.dpr`.

What you will build

The smallest pivot there is. Thirteen sales records, each with a region, a product and an amount, are loaded in code. The grid puts the regions on the rows, the products on the columns and the sum of the sales in the cells, and adds a Grand Total row and column on its own.



Product	Keyboard	Laptop	Monitor	Grand Total
Region	Sum of Sales	Sum of Sales	Sum of Sales	Sum of Sales
East	50	2.400	310	2.760
North	45	2.350	320	2.715
South	90	1.300	640	2.030
West	135	1.250	960	2.345
Grand Total	320	7.300	2.230	9.850

Figure 4.1. Demo 01. The header of the values reads “Sum of Sales”, a caption the grid builds because the value field has no Caption.

Components on the form

Component	Class	Properties you set
`Grid`	`TsgcVCLPivotGrid`	`Align = alClient`. `Pivot.Fields`: `Region` on the rows, `Product` on the columns (`Area = pvaColumn`), `Sales` as the value (`Area = pvaData`, `DisplayFormat = '#,##0'`).

One component, three fields. Region has no `Area` line in the form file because `pvaRow` is the default, and Sales has no `Aggregation` line because `pvfSum` is the default:

```
object Grid: TsgcVCLPivotGrid
  Left = 0
  Top = 0
  Width = 520
  Height = 220
  Align = alClient
  TabOrder = 0
  Pivot.Fields = <
    item
      FieldName = 'Region'
      AreaIndex = 0
    end
    item
      FieldName = 'Product'
      Area = pvaColumn
      AreaIndex = 0
    end
    item
      FieldName = 'Sales'
      Area = pvaData
      AreaIndex = 0
      DisplayFormat = '#,##0'
    end>
end
```

The code

The `uses` clause names the grid unit, the engine unit and the unit of the types such as `pvtText`:

```
uses
  Classes, Controls, Forms,
  // sgc
  sgcPivotGrid_Types, sgcPivotGrid_Engine, sgcPivotGrid_VCL;
```

And this is the only event handler of the demo:

```
procedure TfrmHelloPivot.FormCreate(Sender: TObject);
begin
  { BeginLoad and EndLoad bracket the load, so the pivot is calculated once,
    at EndLoad, and not after every record. }
  Grid.Pivot.BeginLoad;
  try
    { First the columns of the source data: a name and a type. The names
      are the ones used by FieldName in Grid.Pivot.Fields. }
    Grid.Pivot.DefineColumn('Region', pvtText);
    Grid.Pivot.DefineColumn('Product', pvtText);
    Grid.Pivot.DefineColumn('Sales', pvtFloat);
    { Then one AppendRow per record, the values in the order of the
      columns above. }
    Grid.Pivot.AppendRow(['North', 'Laptop', 1200]);
    Grid.Pivot.AppendRow(['North', 'Monitor', 320]);
    Grid.Pivot.AppendRow(['North', 'Laptop', 1150]);
    Grid.Pivot.AppendRow(['North', 'Keyboard', 45]);
    Grid.Pivot.AppendRow(['South', 'Laptop', 1300]);
    // ...
    Grid.Pivot.AppendRow(['West', 'Keyboard', 135]);
  finally
    Grid.Pivot.EndLoad;
  end;
end;
```

How it works

1. The form loads from its DFM. The grid creates its engine and reads the three fields of `Pivot.Fields`. There is no data yet, so the grid is empty.
2. `FormCreate` calls `BeginLoad`. From now on the engine collects records without recalculating.
3. The three `DefineColumn` calls create the columns `Region`, `Product` and `Sales`. These are the names the fields refer to.
4. The thirteen `AppendRow` calls add the records.
5. `EndLoad` matches the field names with the columns, groups the records by region and product, sums the sales of each group, adds the totals and repaints the grid.

Try this

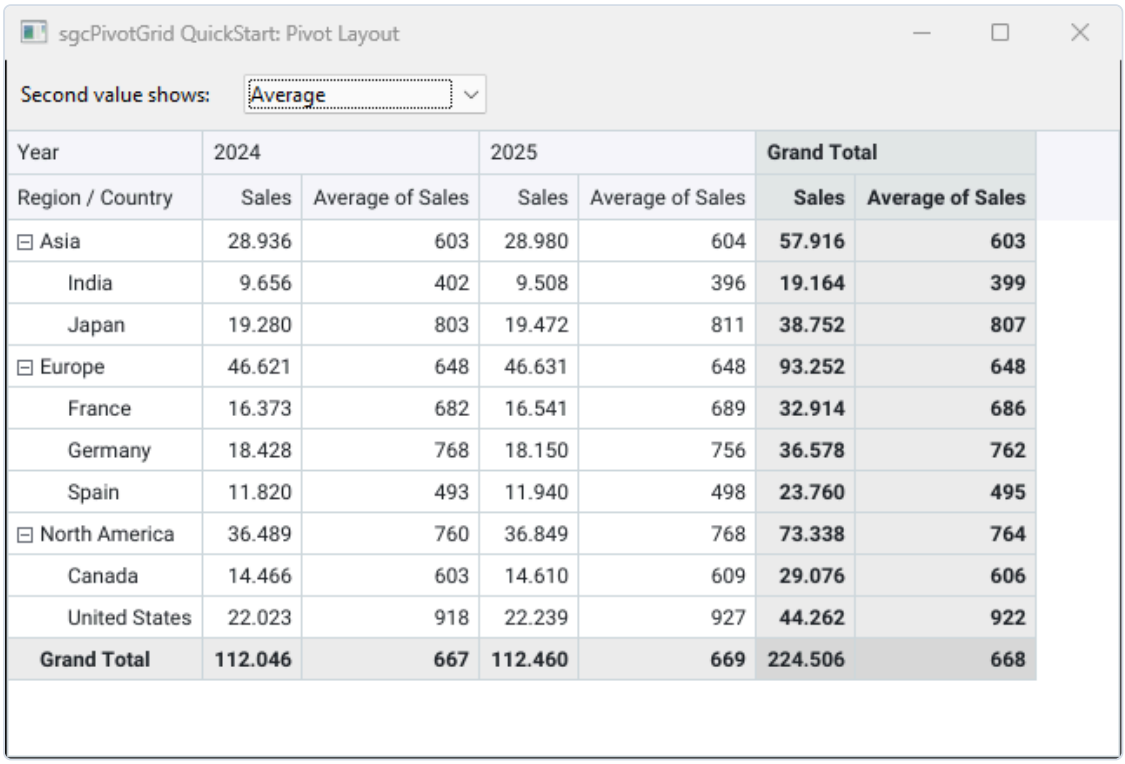
- Swap the `Area` of the Region and Product fields in the collection editor of `Pivot.Fields`. Run again: the products are now the rows.
- Set `Aggregation = pvfCount` on the Sales field. The cells now count the sales, and the header changes to match.
- Set `Grid.Theme` to `utDark`.

4.2 Demo 02: Pivot Layout

Folder `02.PivotLayout`, project `PivotLayout.dpr`.

What you will build

A pivot with a real layout: two levels on the rows (regions, and the countries inside them), years on the columns derived from a date column, and two values side by side, the total sales and a second value that the user switches between average, sum, count, minimum and maximum with a combo box.



Year	2024		2025		Grand Total	
Region / Country	Sales	Average of Sales	Sales	Average of Sales	Sales	Average of Sales
⊟ Asia	28.936	603	28.980	604	57.916	603
India	9.656	402	9.508	396	19.164	399
Japan	19.280	803	19.472	811	38.752	807
⊟ Europe	46.621	648	46.631	648	93.252	648
France	16.373	682	16.541	689	32.914	686
Germany	18.428	768	18.150	756	36.578	762
Spain	11.820	493	11.940	498	23.760	495
⊟ North America	36.489	760	36.849	768	73.338	764
Canada	14.466	603	14.610	609	29.076	606
United States	22.023	918	22.239	927	44.262	922
Grand Total	**112.046**	**667**	**112.460**	**669**	**224.506**	**668**

Figure 4.2. Demo 02. Each region row is the subtotal of its countries, and each year has two value columns.

Components on the form

Component	Class	Properties you set
`pnlTop`	`TPanel`	`Align = alTop`. Holds the label and the combo box.
`lblValue`	`TLabel`	`Caption = 'Second value shows:'`
`cboAggregation`	`TComboBox`	`Style = csDropDownList`, items Sum, Count, Average, Min, Max, `ItemIndex = 2`, `OnChange = cboAggregationChange`
`Grid`	`TsgcVCLPivotGrid`	`Align = alClient`, five fields in `Pivot.Fields` (below)

```
Pivot.Fields = <
  item
    FieldName = 'Region'
    AreaIndex = 0
  end
  item
    FieldName = 'Country'
    AreaIndex = 1
  end
  item
```

```
        FieldName = 'Date'
        Caption = 'Year'
        Area = pvaColumn
        AreaIndex = 0
        DateGrouping = pvdYear
      end
      item
        FieldName = 'Sales'
        Caption = 'Sales'
        Area = pvaData
        AreaIndex = 0
        DisplayFormat = '#,##0'
      end
      item
        FieldName = 'Sales'
        Area = pvaData
        AreaIndex = 1
        DisplayFormat = '#,##0'
        Aggregation = pvfAverage
      end>
```

Read the layout as a sentence: rows are Region, then Country inside it (`AreaIndex` 0 and 1); columns are the year of the Date; values are the sum of Sales and the average of Sales. The second value field has no `Caption`, so its header follows its aggregation.

The code

The demo generates two years of sales with a fixed formula, so every run shows the same numbers. Note the date column: the records carry full dates, and the Year field groups them.

```
procedure TfrmPivotLayout.DoLoadSales;
var
  vYear, vMonth, vCountry, vSale: Integer;
  vAmount: Double;
begin
  { Two years of sales: one to three sales per country and month. The
    amounts follow a fixed formula, so every run shows the same numbers. }
  Grid.Pivot.BeginLoad;
  try
    Grid.Pivot.DefineColumn('Region', pvtText);
    Grid.Pivot.DefineColumn('Country', pvtText);
    Grid.Pivot.DefineColumn('Date', pvtDateTime);
    Grid.Pivot.DefineColumn('Quantity', pvtInteger);
    Grid.Pivot.DefineColumn('Sales', pvtFloat);
    for vYear := 2024 to 2025 do
      for vMonth := 1 to 12 do
        for vCountry := Low(CS_COUNTRIES) to High(CS_COUNTRIES) do
          for vSale := 0 to (vMonth + vCountry) mod 3 do
          begin
            vAmount := CS_AMOUNTS[vCountry] *
              (70 + (vMonth * 17 + vCountry * 29 + vSale * 11 + vYear) mod 61)
              / 100;
            Grid.Pivot.AppendRow([CS_REGIONS[vCountry], CS_COUNTRIES[vCountry],
              EncodeDate(vYear, vMonth, 1 + (vCountry * 3 + vSale * 7) mod 28),
              1 + (vMonth + vCountry + vSale) mod 9, vAmount]);
          end;
  finally
    Grid.Pivot.EndLoad;
```

```
  end;
end;
```

The combo box changes the aggregation of the second value field. No refresh call is needed:

```
procedure TfrmPivotLayout.cboAggregationChange(Sender: TObject);
var
  vField: TsgcPivotField;
begin
  { FieldAt returns the field of an area at an AreaIndex: here the second
    data field. The items of the combo box follow the order of
    TsgcPivotAggregation: pvfSum, pvfCount, pvfAverage, pvfMin, pvfMax. }
  vField := Grid.Pivot.Fields.FieldAt(pvaData, 1);
  if vField <> nil then
    vField.Aggregation := TsgcPivotAggregation(cboAggregation.ItemIndex);
end;
```

How it works

1. `FormCreate` calls `DoLoadSales`, which loads five columns. The layout uses four of them; Quantity is loaded but not used yet.
2. Because Region has `AreaIndex = 0` and Country `AreaIndex = 1`, the countries are nested inside their region, and each region row shows the subtotal of its countries.
3. `DateGrouping = pvdYear` turns every date into its year, so the columns are 2024 and 2025 rather than hundreds of single days.
4. The Sales column is used by two value fields. Each column item gets one data column per value field, plus the Grand Total.
5. When the user picks another entry in the combo box, `FieldAt(pvaData, 1)` finds the second value field and its `Aggregation` changes. With `AutoRecalculate` on, the engine recalculates and the header caption follows.

Try this

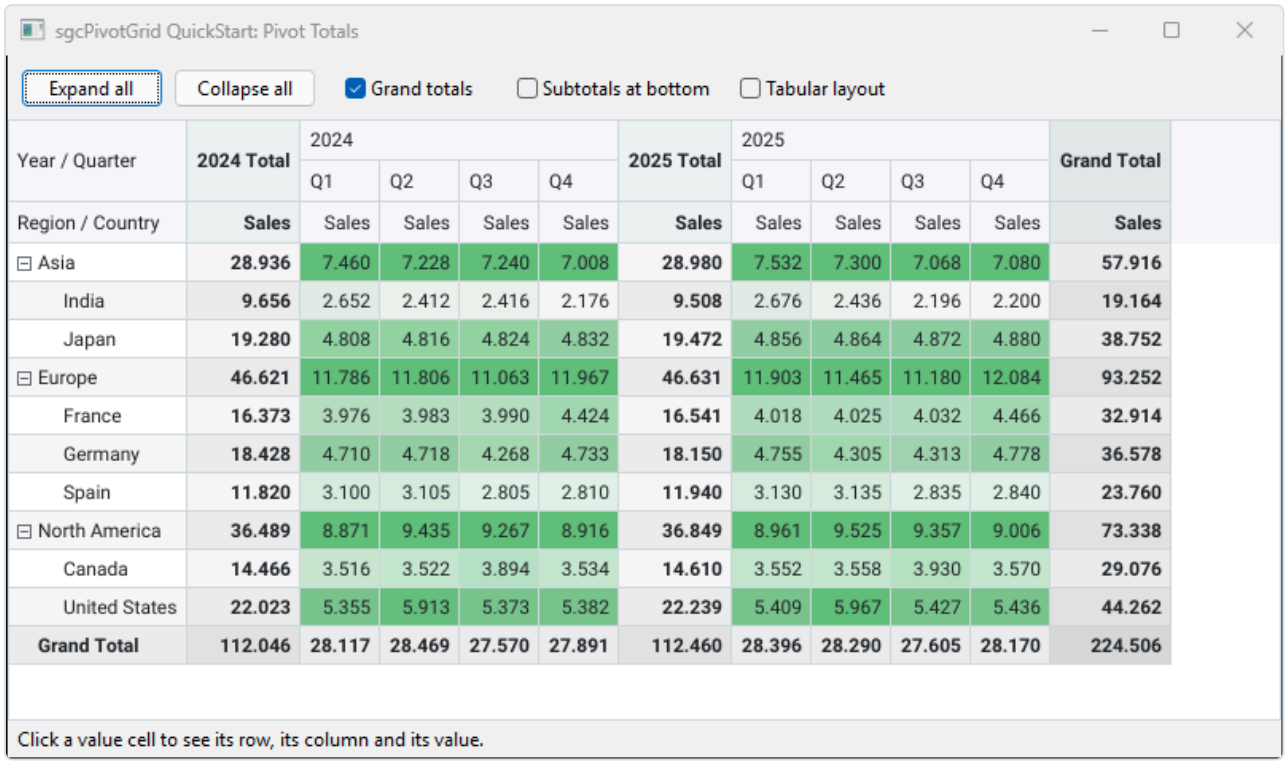
- Add a sixth field with `FieldName = 'Quantity'`, `Area = pvaData` and `AreaIndex = 2`. The column is already in the data.
- Set `DateGrouping = pvdQuarter` or `pvdMonth` on the Date field, and change its `Caption` to match.
- Set `SortBy = pvsValue` and `SortOrder = pvoDescending` on the Country field: inside each region, the best selling country comes first.

4.3 Demo 03: Pivot Totals

Folder `03.PivotTotals`, project `PivotTotals.dpr`.

What you will build

The same two years of sales as demo 02, now with quarters nested inside the years. Buttons expand and collapse everything, check boxes switch the grand totals, the position of the subtotals and the row layout, a colour scale shades the values from white to green, and a click on a value shows its row, its column and its value in a status line.



Year / Quarter	2024 Total	2024				2025 Total	2025				Grand Total
		Q1	Q2	Q3	Q4		Q1	Q2	Q3	Q4	
Region / Country	Sales	Sales	Sales	Sales	Sales	Sales	Sales	Sales	Sales	Sales	Sales
⊟ Asia	28.936	7.460	7.228	7.240	7.008	28.980	7.532	7.300	7.068	7.080	57.916
India	9.656	2.652	2.412	2.416	2.176	9.508	2.676	2.436	2.196	2.200	19.164
Japan	19.280	4.808	4.816	4.824	4.832	19.472	4.856	4.864	4.872	4.880	38.752
⊟ Europe	46.621	11.786	11.806	11.063	11.967	46.631	11.903	11.465	11.180	12.084	93.252
France	16.373	3.976	3.983	3.990	4.424	16.541	4.018	4.025	4.032	4.466	32.914
Germany	18.428	4.710	4.718	4.268	4.733	18.150	4.755	4.305	4.313	4.778	36.578
Spain	11.820	3.100	3.105	2.805	2.810	11.940	3.130	3.135	2.835	2.840	23.760
⊟ North America	36.489	8.871	9.435	9.267	8.916	36.849	8.961	9.525	9.357	9.006	73.338
Canada	14.466	3.516	3.522	3.894	3.534	14.610	3.552	3.558	3.930	3.570	29.076
United States	22.023	5.355	5.913	5.373	5.382	22.239	5.409	5.967	5.427	5.436	44.262
Grand Total	112.046	28.117	28.469	27.570	27.891	112.460	28.396	28.290	27.605	28.170	224.506

Figure 4.3. Demo 03. Each year shows its quarters and a year total column; the colour scale shades the cells by value.

Components on the form

Component	Class	Properties you set
`btnExpand`, `btnCollapse`	`TButton`	`Caption` Expand all and Collapse all, `OnClick`
`chkGrandTotals`	`TCheckBox`	`Checked = True`, `OnClick = chkGrandTotalsClick`
`chkSubtotalsBottom`, `chkTabular`	`TCheckBox`	`OnClick`
`lblCell`	`TLabel`	`Align = alBottom`, shows the clicked cell
`Grid`	`TsgcVCLPivotGrid`	`Pivot.Fields`: Region and Country on the rows; Date twice on the columns, `Caption = 'Year'` with `DateGrouping = pvdYear` at `AreaIndex` 0 and `Caption = 'Quarter'` with `pvdQuarter` at `AreaIndex` 1; Sales as the value. `AlternateRows = True`, one format rule, `OnCellClick = GridCellClick`.

The format rule is a colour scale on the first value field (`Measure = 0`). `MinColor` keeps its default 0, transparent, and `MaxColor` is the green `$FF63BE7B`. The form file stores colours as signed integers, which is why the green appears as a negative number:

```
    AlternateRows = True
    FormatRules = <
      item
        Kind = pvrColorScale
        Measure = 0
        MaxColor = -10240389
      end>
    OnCellClick = GridCellClick
```

The code

The data is loaded by the same `DoLoadSales` as in demo 02. The buttons and check boxes are one or two lines each. `pvxRows` and `pvxColumns` name the two axes; the grand totals and the subtotal position are engine properties, the row layout is a grid property:

```
procedure TfrmPivotTotals.btnExpandClick(Sender: TObject);
begin
  // pvxRows is the row axis, pvxColumns the column axis
  Grid.Pivot.ExpandAll(pvxColumns);
  Grid.Pivot.ExpandAll(pvxRows);
end;

procedure TfrmPivotTotals.btnCollapseClick(Sender: TObject);
begin
  Grid.Pivot.CollapseAll(pvxColumns);
  Grid.Pivot.CollapseAll(pvxRows);
end;

procedure TfrmPivotTotals.chkGrandTotalsClick(Sender: TObject);
begin
  Grid.Pivot.RowGrandTotals := chkGrandTotals.Checked;
  Grid.Pivot.ColumnGrandTotals := chkGrandTotals.Checked;
end;

procedure TfrmPivotTotals.chkSubtotalsBottomClick(Sender: TObject);
begin
  if chkSubtotalsBottom.Checked then
    Grid.Pivot.SubtotalPosition := pvpBottom
  else
    Grid.Pivot.SubtotalPosition := pvpTop;
end;

procedure TfrmPivotTotals.chkTabularClick(Sender: TObject);
begin
  if chkTabular.Checked then
    Grid.RowLayout := pvlTabular
  else
    Grid.RowLayout := pvlCompact;
end;
```

The click handler reads the calculated result. A small helper turns a row or column item into its caption:

```
{ The caption of a row or column item of the result: the labels of its
  members joined by ' / ' (for example '2024 / Q2'), or 'Grand Total'. }
function DoItemCaption(aResult: TsgcPivotResult; aAxis: TsgcPivotAxisKind;
```

```
  aIndex: Integer): string;
var
  vItem: TsgcPivotItem;
begin
  vItem := aResult.Item(aAxis, aIndex);
  if vItem.Kind = pvkGrandTotal then
    Result := 'Grand Total'
  else
    Result := aResult.NodePath(aAxis, vItem.Node);
end;

procedure TfrmPivotTotals.GridCellClick(Sender: TObject;
  const aHit: TsgcPivotHitInfo);
var
  vResult: TsgcPivotResult;
  vText: string;
begin
  { aHit tells which part of the grid was clicked. For a value cell (Part =
    pvhCell) RowItem is the row and DataCol the column of the value. }
  if aHit.Part <> pvhCell then
    Exit;
  vText := '';
  { The calculated result is read between Lock and Unlock. A data column is
    a column item times the number of values (MeasureCount). }
  Grid.Pivot.Lock;
  try
    vResult := Grid.Pivot.Result;
    if vResult <> nil then
      vText := DoItemCaption(vResult, pvxRows, aHit.RowItem) + ', ' +
        DoItemCaption(vResult, pvxColumns, aHit.DataCol div
        vResult.MeasureCount) + ': ' + vResult.CellText(aHit.RowItem,
        aHit.DataCol);
  finally
    Grid.Pivot.Unlock;
  end;
  lblCell.Caption := vText;
end;
```

The units of this demo add `sgcPivotGrid_Result` (`TsgcPivotResult`, `TsgcPivotItem`) and `sgcPivotGrid_Render` (`TsgcPivotHitInfo`, `pvhCell`) to the ones of demo 02.

How it works

1. Two fields use the Date column on the columns. The Year field is the outer level, the Quarter field the inner one, so every year shows its four quarters plus a year total.
2. **Expand all** and **Collapse all** change only the expanded state of the members. The engine rebuilds the visible rows and columns from the data it already aggregated, so the change is instant.
3. The check boxes set engine properties (`RowGrandTotals`, `ColumnGrandTotals`, `SubtotalPosition`) or a grid property (`RowLayout`), and the grid follows by itself.
4. The colour scale computes its range from the values and shades every value cell between transparent and green.
5. A click raises `OnCellClick` with a `TsgcPivotHitInfo`. For a value cell, `RowItem` and `DataCol` locate it in the result. The handler locks the engine, reads the captions and the cell text, unlocks it and only then updates the label.

Try this

- Set `Expanded = False` on the **Year** field (not on the Quarter field). The years start closed and show only their totals; the plus buttons open them.
- Add `Grid.Pivot.GrandTotalCaption := 'Total';` at the end of `FormCreate`.
- Add a second format rule with `Kind = pvrValue`, `Operator = pvcLess`, `Value = 3000` and `Bold = True`. Try `BackColor` and `ForeColor` as well, remembering that the first byte of an ARGB colour is the opacity.

4.4 Demo 04: Pivot DataSet

Folder `04.PivotDataSet`, project `PivotDataSet.dpr`.

What you will build

A form with forty orders in a `TDBGrid` at the top and a pivot of the same orders below it. The pivot is fed by a `TsgcPivotSource_DataSet`, so you write no loading code at all. Edit an amount, insert or delete an order with the navigator, post it, and the pivot follows at once.

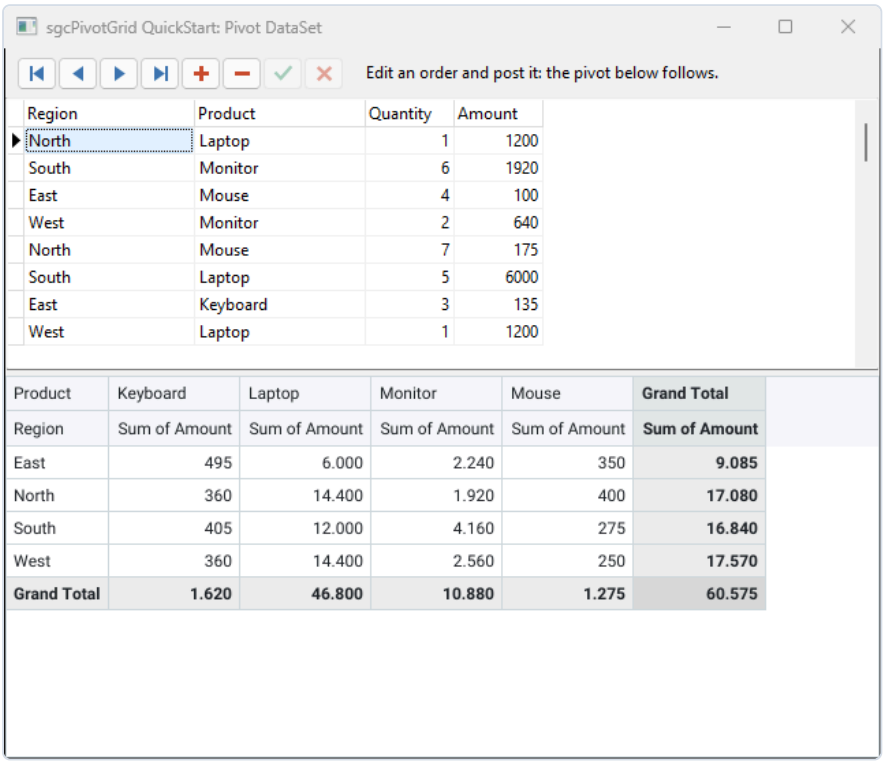


Figure 4.4. Demo 04. The DBGrid shows the records; the pivot below shows their summary.

Components on the form

Component	Class	Properties you set
`cdsSales`	`TClientDataSet`	`FieldDefs`: Region and Product (`ftString`, 20), Quantity (`ftInteger`), Amount (`ftFloat`). `StoreDefs = True`.
`dsSales`	`TDataSource`	`DataSet = cdsSales`
`Navigator`	`TDBNavigator`	`DataSource = dsSales`, without the Edit and Refresh buttons
`DBGrid`	`TDBGrid`	`Align = alTop`, `DataSource = dsSales`
`Grid`	`TsgcVCLPivotGrid`	`Align = alClient`. `Pivot.Fields`: Region on the rows, Product on the columns, Amount as the value with `DisplayFormat = '#,##0'`.
`PivotSource`	`TsgcPivotSource_DataSet`	`DataSet = cdsSales`, `PivotGrid = Grid`, `Active = True`

The source component is the whole connection. `LiveUpdates` does not appear in the form file because `True` is its default:

```
  object PivotSource: TsgcPivotSource_DataSet
    Active = True
    DataSet = cdsSales
    PivotGrid = Grid
    Left = 560
    Top = 192
  end
```

The code

The `uses` clause adds the database units and `MidasLib`, which links the `TClientDataSet` engine into the executable so no `midas.dll` is needed:

```
uses
  Classes, Controls, Forms, StdCtrls, ExtCtrls, Grids, DBGrids, DBCtrls,
  DB, DBClient, MidasLib,
  // sgc
  sgcPivotGrid_Engine, sgcPivotGrid_VCL, sgcPivotGrid_Source_DataSet;
```

The code only creates the in-memory dataset and fills it with orders. There is no pivot code in the unit:

```
procedure TfrmPivotDataSet.FormCreate(Sender: TObject);
begin
  { An in-memory TClientDataSet is created from its FieldDefs. A query on a
    real database would just be opened. }
  cdsSales.CreateDataSet;
  DoAddOrders;
end;

procedure TfrmPivotDataSet.DoAddOrders;
var
  i, vRegion, vProduct, vQuantity: Integer;
begin
  { DisableControls stops the notifications while the orders are added, so
    the pivot loads once, at EnableControls, instead of once per order. }
  cdsSales.DisableControls;
  try
    for i := 0 to 39 do
    begin
      vRegion := i mod 4;
      vProduct := (i * 7 div 4) mod 4;
      vQuantity := 1 + (i * 5) mod 7;
      cdsSales.AppendRecord([CS_REGIONS[vRegion], CS_PRODUCTS[vProduct],
        vQuantity, vQuantity * CS_PRICES[vProduct]]);
    end;
    cdsSales.First;
  finally
    cdsSales.EnableControls;
  end;
end;
```

How it works

1. The source is `Active`, so it loads the dataset as soon as the dataset opens. `CreateDataSet` opens it empty.
2. `DisableControls` silences the dataset while forty records are appended. Without it, the source would reload the pivot after every single record.

3. `EnableControls` lets the notifications through again, and the source reloads the pivot once. Each dataset field becomes a column with the same name and a matching type, so the fields Region, Product and Amount of the layout find their data.
4. From now on every posted change reaches the pivot: an edit once it is posted, an insert, a delete, a refresh. Moving the cursor with Next or Prior does not reload.

> **Tip.** The same three properties connect a FireDAC `TFDQuery`, an ADO query or any other `TDataSet`. Only the dataset changes; the pivot side stays the same.

Try this

- Add a button that filters the dataset: `cdsSales.Filter := 'Region = ''North''';` then `cdsSales.Filtered := True;`. A filter change reloads the pivot, which now shows one region.
- Set `PivotSource.LiveUpdates := False` and add a button that calls `PivotSource.Refresh`. Now the pivot changes only when you ask.
- Add Quantity as a second value field (`Area = pvaData`, `AreaIndex = 1`).

4.5 Demo 05: Pivot Sales App

Folder `05.PivotSalesApp`, project `PivotSalesApp.dpr`.

What you will build

A small but complete sales analysis window. Three thousand sales are loaded in code. A field list on the left lets the user drag fields between the filters, the columns, the rows and the values. A Channel filter field restricts the sales, the years start collapsed, a double click on a value tells how many sales are behind it, and two buttons export the current view to CSV and HTML.

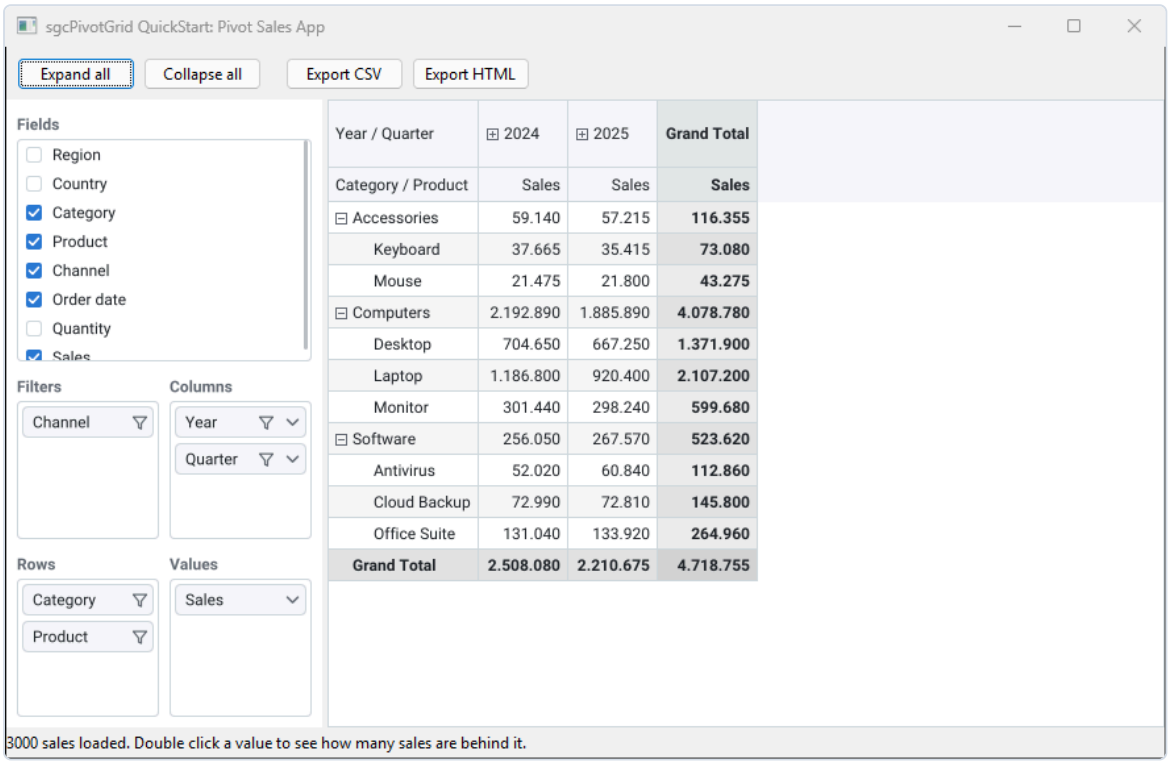


Figure 4.5. Demo 05. The field list shows the layout of the grid: Channel in Filters, Year and Quarter in Columns, Category and Product in Rows, Sales in Values.

Components on the form

Component	Class	Properties you set
`btnExpand`, `btnCollapse`, `btnCSV`, `btnHTML`	`TButton`	`Caption`, `OnClick`
`FieldList`	`TsgcVCLPivotFieldList`	`Align = alLeft`, `PivotGrid = Grid`, `DefaultDateGrouping = pvdYear`
`Grid`	`TsgcVCLPivotGrid`	`Align = alClient`, six fields (below), `AlternateRows = True`, `OnDrillDown = GridDrillDown`
`lblStatus`	`TLabel`	`Align = alBottom`
`dlgSave`	`TSaveDialog`	Asks for the export file name

The layout has two new things: a field in the filter area, and `Expanded = False` on the Year field, so the years start closed and the quarters stay hidden until the user opens a year:

```
      item
        FieldName = 'Date'
        Caption = 'Year'
        Area = pvaColumn
        AreaIndex = 0
        DateGrouping = pvdYear
        Expanded = False
      end
      item
        FieldName = 'Date'
        Caption = 'Quarter'
        Area = pvaColumn
        AreaIndex = 1
        DateGrouping = pvdQuarter
      end
      item
        FieldName = 'Channel'
        Area = pvaFilter
        AreaIndex = 0
      end
```

Category and Product are on the rows and Sales is the value, as in the earlier demos.

The code

The load is the familiar bracket, with eight columns. Look at the Date column: the third parameter of `DefineColumn` is the caption that the field list shows for it.

```
procedure TfrmPivotSalesApp.DoLoadSales;
var
  i, vCountry, vProduct, vQuantity: Integer;
begin
  FSeed := 2026;
  Grid.Pivot.BeginLoad;
  try
    { the third DefineColumn parameter is the caption shown in the field
      list and in the headers }
    Grid.Pivot.DefineColumn('Region', pvtText);
    Grid.Pivot.DefineColumn('Country', pvtText);
    Grid.Pivot.DefineColumn('Category', pvtText);
    Grid.Pivot.DefineColumn('Product', pvtText);
    Grid.Pivot.DefineColumn('Channel', pvtText);
    Grid.Pivot.DefineColumn('Date', pvtDateTime, 'Order date');
    Grid.Pivot.DefineColumn('Quantity', pvtInteger);
    Grid.Pivot.DefineColumn('Sales', pvtFloat);
    for i := 1 to CS_ROWS do
    begin
      vCountry := DoRandom(Length(CS_COUNTRIES));
      vProduct := DoRandom(Length(CS_PRODUCTS));
      vQuantity := 1 + DoRandom(8);
      Grid.Pivot.AppendRow([CS_REGIONS[vCountry], CS_COUNTRIES[vCountry],
        CS_CATEGORIES[vProduct], CS_PRODUCTS[vProduct],
        CS_CHANNELS[DoRandom(Length(CS_CHANNELS))],
        EncodeDate(2024, 1, 1) + DoRandom(731), vQuantity,
        vQuantity * CS_PRICES[vProduct]]);
    end;
  finally
    Grid.Pivot.EndLoad;
  end;
  lblStatus.Caption := Format('%d sales loaded. Double click a value to ' +
```

```
    'see how many sales are behind it.', [Grid.Pivot.SourceRowCount]);
end;
```

The exports are one call each. The CSV button is shown here; the HTML button is the same with `SaveToHTML`:

```
procedure TfrmPivotSalesApp.btnCSVClick(Sender: TObject);
begin
  { The exports write the current view: the visible rows and columns, with
    the totals, in UTF-8. }
  dlgSave.Filter := 'CSV files (*.csv)|*.csv';
  dlgSave.DefaultExt := 'csv';
  dlgSave.FileName := 'sales.csv';
  if dlgSave.Execute then
  begin
    Grid.SaveToCSV(dlgSave.FileName);
    lblStatus.Caption := 'Saved ' + dlgSave.FileName;
  end;
end;
```

A double click on a value cell raises `OnDrillDown`. The handler asks the engine which source records are behind the cell:

```
procedure TfrmPivotSalesApp.GridDrillDown(Sender: TObject;
  aRowItem, aDataCol: Integer; var aHandled: Boolean);
var
  vRows: TsgcPivotIntegers;
  vCount: Integer;
begin
  { GetSourceRows fills vRows with the indexes of the source rows behind a
    value (Pivot.SourceValue reads their columns). The filter of the
    Channel field applies here too. }
  vCount := Grid.Pivot.GetSourceRows(aRowItem, aDataCol, vRows);
  lblStatus.Caption := Format('%d sales are behind this value.', [vCount]);
  aHandled := True;
end;
```

How it works

1. The field list is bound to the grid with `PivotGrid = Grid`. It lists every loaded column at the top (with the caption Order date for the Date column) and shows the current layout in four boxes: Filters, Columns, Rows and Values.
2. The user drags a field from the list into a box, between boxes or out of them, and the grid recalculates. A date column dropped from the list on Filters, Columns or Rows gets the `DefaultDateGrouping` of the field list, here years.
3. The filter glyph on a field opens a checklist of its members. Unchecking Partner and Retail on the Channel field, for example, leaves only the Online sales in every cell and total. The arrow on a value field opens a menu with the aggregations.
4. `Expanded = False` on the Year field makes the years start closed. The plus button on a year shows its quarters.
5. A double click on a value calls `GridDrillDown`. `GetSourceRows` fills an array with the indexes of the records behind the value, filters included, and returns their count. `Pivot.SourceValue(row, column)` would read their values, for example to fill a detail list.
6. `SaveToCSV` and `SaveToHTML` write what the grid shows: the visible rows and columns, with their totals, in UTF-8. Collapse the years first and the file holds only the year totals.

Try this

- Set `FilterValues` of the Channel field to `Online` in the Object Inspector, so the window starts filtered.
- Set `FieldList.DeferUpdate := True` and add a button that calls `FieldList.ApplyChanges`. The user can now rearrange several fields and recalculate once, which matters with large data.
- Add a button that calls `Grid.SaveToJSON(FileName)` or `Grid.SaveToPNG(FileName)`. The grid also has `SaveToSVG`, `Print`, `ShowPrintPreview` and `ExportToPDF`.

5. Main components

The SGC PivotGrid palette page is short, and three of its components cover a typical first application: the grid, the field list and the dataset source. This chapter introduces each of them, then the engine classes you meet in code, and finally the components you will want later.

Every class has its own help topic, named after the class, such as `TsgcVCLPivotGrid.htm` or `TsgcPivotField.htm`. Each topic lists the properties, methods and events with their types and defaults.

Reference table

Class	Palette page	What it is for	Key members
`TsgcVCLPivotGrid`	SGC PivotGrid	The VCL pivot grid	`Pivot`, `Theme`, `RowLayout`, `FormatRules`, `OnCellClick`, `OnDrillDown`, `SaveToCSV`
`TsgcFMXPivotGrid`	SGC PivotGrid (FireMonkey)	The FireMonkey pivot grid	The same as the VCL grid
`TsgcVCLPivotFieldList`	SGC PivotGrid	Drag and drop layout editor for the user	`PivotGrid`, `DefaultDateGrouping`, `DeferUpdate`, `ShowFilterArea`, `ApplyChanges`
`TsgcFMXPivotFieldList`	SGC PivotGrid (FireMonkey)	The FireMonkey field list	The same as the VCL field list
`TsgcPivotSource_DataSet`	SGC PivotGrid	Loads a `TDataSet`	`DataSet`, `PivotGrid`, `Active`, `LiveUpdates`, `Columns`, `Refresh`
`TsgcPivotSource_Table`	SGC PivotGrid	Loads any sgcUI table, such as an sgcGrid	`Table`, `PivotGrid`, `Active`, `LiveUpdates`, `Columns`
`TsgcPivotGridAI`	SGC PivotGrid	A pivot layout from a plain language question	`PivotGrid`, `Provider`, `Active`, `OnLayout`
`TsgcPivotChartTable`	SGC PivotGrid	The pivot result as a table for sgcCharts (Registered editions)	`PivotGrid`, `RowTotals`, `ColumnTotals`
`TsgcPivotEngine`	none, it is `Grid.Pivot`	Data, layout, calculation and result	`BeginLoad`, `DefineColumn`, `AppendRow`, `EndLoad`, `Fields`, `Result`, `Lock`
`TsgcPivotField`	none, an item of `Pivot.Fields`	One field of the layout	`FieldName`, `Area`, `AreaIndex`, `Aggregation`, `DateGrouping`, `Expanded`
`TsgcPivotFormatRule`	none, an item of `FormatRules`	Conditional format of the values	`Kind`, `Measure`, `Operator`, `Value`, `MinColor`, `MaxColor`

The pivot grid

TsgcVCLPivotGrid

The control you drop first. It owns its engine, published as `Pivot`, so the layout and the totals are edited under `Pivot` in the Object Inspector, while the grid itself holds the look and the interaction.

- **Look:** `Theme`, `AccentColor`, `FontSize`, `RowHeight`, `HeaderHeight`, `IndentWidth`, `GridLines`, `AlternateRows`, `TotalsBold`, `ShowExpandButtons`, `RowLayout` (`pvlCompact` or `pvlTabular`).
- **Columns:** `AutoColumnWidths`, `MinColumnWidth`, `MaxColumnWidth`. The user can also drag the right edge of a header.
- **Values:** `FormatRules` for colours without code; `OnGetCellStyle` and `OnGetCellText` when you need code.
- **Events:** `OnCellClick`, `OnCellDblClick`, `OnDrillDown` (double click or Enter on a value), `OnExpandChange`, `OnSelectionChange`.
- **Output:** `SaveToCSV`, `SaveToHTML`, `SaveToJSON`, `SaveToPNG`, `SaveToSVG`, `Print`, `ShowPrintPreview` and `ExportToPDF`, with `PrintSettings` for the page.

The keyboard works as in a spreadsheet: arrows, Page Up and Page Down, Home and End, + and − to expand and collapse the selected row, Enter to drill down on a value.

TsgcFMXPivotGrid

The FireMonkey twin, from Delphi XE6 on, with the same engine and the same properties, drawn by the same renderer. It adds touch pan for mobile platforms. Everything in this guide applies to it; only the class name and the unit (`sgcPivotGrid_FMX`) change.

The field list

TsgcVCLPivotFieldList

Gives the user the layout. Set `PivotGrid` and it lists the loaded columns at the top, with four boxes below: Filters, Columns, Rows and Values. Dragging a column into a box adds a field, dragging a field between boxes moves it, and dragging it out removes it. The filter glyph of a field opens a checklist of its members, and a value field has a menu with the aggregations.

- `DefaultDateGrouping`: the grouping given to a date column dropped on Filters, Columns or Rows. The default is `pvdNone`; demo 05 uses `pvdYear`.
- `DeferUpdate`: collect the changes and apply them only when you call `ApplyChanges`, so a large pivot recalculates once.
- `ShowFilterArea`: show or hide the Filters box.
- `OnBeforeFieldMove` and `OnFieldMoved`: react to the user, or refuse a move.

The data sources

TsgcPivotSource_DataSet

Connects any `TDataSet` to a pivot grid. `DataSet`, `PivotGrid` and `Active` are all it needs. `LiveUpdates` (on by default) reloads after posts, inserts, deletes, refreshes and filter changes; `Columns` limits the load to some fields; `UseDisplayLabels` uses the `DisplayLabel` of each field as its caption; `Throttle` (milliseconds) gathers quick bursts of changes into one reload. `Load` and `Refresh` reload in code.

TsgcPivotSource_Table

The same idea for sgcUI tables. Point `Table` at a component that exposes an sgcUI table, for example an sgcGrid, and `PivotGrid` at the pivot. With `LiveUpdates`, the pivot reloads when the table changes. The `GridToPivotToChart` demo uses it.

Your own code

No component at all: `BeginLoad`, `DefineColumn`, `AppendRow` and `EndLoad` on `Grid.Pivot`, as in demos 01, 02, 03 and 05. This is the right choice for data from files, web services or your own objects.

The engine and its fields

TsgcPivotEngine

You reach it as `Grid.Pivot`. Besides loading, layout and totals (chapter 3), it offers `SourceRowCount` and `SourceValue` to read the loaded records, `GetSourceRows` for drill down, `BlankCaption` for the label of empty members, `EmptyText` for cells with no records, `MaxCells` as a safety limit on the size of the result, and the events `OnCalculated` and `OnLayoutChange`. Because it uses no visual framework, it can also be created on its own, for a report in a console application or on a server.

TsgcPivotField

One item of `Pivot.Fields`, described in Fields and the four areas. Beyond the properties used in the demos, `LabelFilter` and `ValueFilter` filter members by their label or by their value, and `SortMeasure` chooses the value field a sort by value uses.

Beyond the basics

- **`TsgcPivotGridAI`** turns a question such as “top 3 countries by sales” into a pivot layout through an AI provider, validates the answer against the fields of the pivot and applies it. `TsgcPivotGridAIQuery`, in the sgcWebSockets bridge, does the same with the AI chat of sgcWebSockets. See the `AskYourPivot` demo.
- **`TsgcPivotChartTable`** hands the pivot result to sgcCharts, for a pivot chart that follows the layout. See the `PivotChart` and `GridToPivotToChart` demos.
- **`TsgcPivotGridRenderer`** draws a pivot without a window, to PNG or SVG. See the `HeadlessReport` demo.
- **The advanced features** of the Registered editions (calculated fields, show values as, Top N, saving and loading the layout, the XLSX export) are shown in the `AdvancedPivot` demo.

6. Common problems

These are the problems that come up most often in a first pivot. Each entry says what you see, why it happens and how to fix it. When the grid shows something unexpected, start by checking two numbers in the debugger: `Grid.Pivot.SourceRowCount` (did the records arrive?) and the names in `Pivot.Fields` (do they match the columns?).

Data and layout

The grid stays empty

What you see. The grid shows no headers or no values, although the form has fields in `Pivot.Fields`.

Why. Either no data arrived, or the fields do not find their columns. `SourceRowCount` is 0 when you never loaded anything, or when a `TsgcPivotSource_DataSet` is not `Active`, has no `DataSet` or `PivotGrid`, or its dataset is closed. A row, column or filter field whose `FieldName` matches no column is silently ignored, and a value field with an unknown name stays empty. A filter that keeps no records also gives an empty pivot.

Fix. Check the spelling of each `FieldName` against the names passed to `DefineColumn`, or the field names of the dataset (case does not matter). In code, `Pivot.SourceColumnName(i)` for `i` from 0 to `Pivot.SourceColumnCount - 1` lists what was loaded. If you set `AutoRecalculate` to `False`, call `Pivot.Recalculate`.

The FieldName drop down list is empty

What you see. In the collection editor, the drop down list of `FieldName` offers nothing.

Why. The list shows the columns the engine knows at design time: the loaded columns, or the columns of a dataset or table source bound to the grid on the same form. When the data is loaded from code at run time, as in demos 01, 02, 03 and 05, there is nothing to list yet.

Fix. Type the column names by hand. That is perfectly fine: the fields find their columns when the data is loaded.

Loading is slow

What you see. Loading a few thousand records takes seconds, or the window flickers while it fills.

Why. `AppendRow` outside a `BeginLoad` and `EndLoad` bracket recalculates the pivot after every row. With a dataset source, appending records one by one while the source listens reloads the pivot after every record.

Fix. Put the whole load inside `BeginLoad` and `EndLoad`, as every demo does. When you fill a dataset in code, wrap it in `DisableControls` and `EnableControls`, as demo 04 does. For a dataset that changes in quick bursts, set `Throttle` on the source.

The pivot does not follow the dataset

What you see. You change a value in the `TDBGrid`, and the pivot keeps the old total.

Why. The source reloads when the dataset reports a data change in browse state: a post, an insert, a delete, a cancel, a refresh or a filter change. An edit that is not posted yet is not a change, and moving with Next, Prior or MoveBy is not either. With `LiveUpdates = False` nothing reloads by itself.

Fix. Post the record (leave the row, or press the Post button of the navigator). If you switched `LiveUpdates` off, call `PivotSource.Refresh` when you want the pivot updated. Note that BLOB, binary and nested fields are never loaded into a pivot.

Dates do not group, or numbers sort as text

What you see. A field with `DateGrouping = pvdYear` shows one member per day, or 10 sorts before 9.

Why. Date grouping works only on a date column, and a numeric sort only on a number column. A column whose type was taken from a string value is a text column.

Fix. Declare the column with the right type, `DefineColumn('Date', pvtDateTime)`, before the first row, and pass real `TDateTime` and numeric values to `AppendRow`, not formatted strings. With a dataset source, use a date or numeric field type in the dataset.

A filter in code shows nothing

What you see. After setting `FilterValues` on a field, the pivot is empty.

Why. `FilterValues` holds member labels, compared case sensitively, and the labels are the ones the grid shows: after the date grouping and the display format of that field. `online` does not match `Online`, and a Year field has labels such as `2024`, not dates.

Fix. Use the labels exactly as they appear in the grid or in the checklist of the field list. `Pivot.GetMemberLabels(Field, List)` returns them in code.

"The pivot layout needs more than ... cells"

What you see. An `EsgcPivotError` with that message when you move a field.

Why. The layout would create more cells than `Pivot.MaxCells` allows. This usually happens when a field with many members, such as a customer or a day, goes on the rows while another goes on the columns.

Fix. Group the dates by month or year, move the large field to the filters, or raise `MaxCells` if you really need that many cells.

Expand, totals and formatting

Expanded = False has no effect

What you see. You set `Expanded = False` on the Quarter field to hide the quarters, and the quarters are still there.

Why. `Expanded` is the starting state of the members *of that field*: whether they show their children. The quarters are children of the years, so it is the years that must start closed.

Fix. Set `Expanded = False` on the outer field, here the Year field, as demo 05 does. Set the Quarter field back to `True`.

A total is not the sum of the cells

What you see. With an average, a minimum or a distinct count, the subtotal or the grand total is not the sum of the values above it.

Why. Totals are calculated from the source records, not from the cells. The total of an average is the average of all the records, and a distinct count counts each value once, however many cells it appears in.

Fix. Nothing to fix: this is the correct result, and the same rule spreadsheets use. If you need a sum, use `pvfSum`.

The colour scale looks flat or too strong

What you see. With collapsed years, every year cell gets the full colour of the scale. In the compact layout the subtotal rows are painted in a strong colour too.

Why. The range of a `pvrColorScale` rule is computed from the cells of the innermost level of the layout, the quarters in demo 03. When the years are collapsed, every year cell is larger than that range, so all of them reach `MaxColor`; in the compact layout the parent rows, such as the regions, come out saturated for the same reason. This is why demo 03 starts fully expanded.

Fix. Use a colour scale with the levels expanded, as demo 03 does, or use value rules (`Kind = pvrValue`) with fixed thresholds instead.

A colour does not show at all

What you see. You set `BackColor` or `MaxColor` to a colour, and nothing is painted.

Why. sgcUI colours are 32 bit ARGB values. The first byte is the opacity, and 0 means fully transparent. A value such as `$63BE7B` has an opacity of 0. Also, VCL `TColor` constants such as `clGreen` store the bytes in a different order and are not ARGB values.

Fix. Write the opacity byte: `$FF63BE7B` for an opaque green. Do not be surprised when the form file shows such a value as a negative number, as in demo 03; that is how it stores values above `$7FFFFFFF`.

The numbers use the wrong separators

What you see. `DisplayFormat = '#,##0'` shows `1.200` on one machine and `1,200` on another.

Why. The display format is applied like `FormatFloat`, so the thousands and decimal separators come from the regional settings of the machine. The screenshots in this guide were taken with a locale that uses dots.

Fix. Usually none: your users see the numbers the way their Windows is configured. If you need a fixed format, change it in `OnGetCellText`.

Empty cells show nothing

What you see. Cells with no records are blank, and users ask whether the value is zero or missing.

Fix. Set `Pivot.EmptyText`, for example to a dash. The grid and the HTML export show it, while `SaveToCSV` writes an empty field and `SaveToJSON` writes null, so the exported data stays numeric.

Editions and compiling

Undeclared identifier CalculatedFields, TopN, ShowAs or SaveLayoutToString

What you see. Code copied from a larger demo or from the help does not compile with the Community edition.

Why. The advanced features (calculated fields, show values as, Top N, layout persistence) are compiled only when `SGC_PIVOTGRID_ADVANCED` is defined, which the Community edition turns off. Those members do not exist there. A form saved with advanced properties set cannot be loaded by a Community build either.

Fix. Use the advanced members only with a Registered or Trial edition, or put them inside `{$IFDEF SGC_PIVOTGRID_ADVANCED}` when the same code must build with both. The QuickStart demos use none of them.

Wrong values or an exception when reading Result

What you see. Code that reads `Pivot.Result` works most of the time but occasionally reads garbage or raises an access violation, especially from a background thread.

Why. Each recalculation publishes a new result and frees the previous one. A reference kept across a recalculation points to freed memory.

Fix. Read the result only between `Pivot.Lock` and `Pivot.Unlock`, and do not keep the `TsgcPivotResult` reference after `Unlock`. Copy what you need into your own variables inside the lock, as demo 03 does with the cell text.

All my fields disappeared

What you see. `Pivot.Fields` is suddenly empty in the designer.

Why. The **Clear Layout** command of the component editor removes every field. It sits right below **Recalculate** in the context menu of the grid.

Fix. Close the form without saving and reopen it, or restore the form file from your version control. Use **Clear Layout** only when you really want to start over.

7. Where next

Once the five QuickStart demos feel familiar, you know the core of sgcPivotGrid. This chapter points you to the larger demos, to the documentation and to support.

The other Delphi demos

The other folders in `Demos\PivotGrid\Delphi` are complete applications. Each one has a `readme.html` that explains what it shows and how to run it.

Demo	What it shows
`SalesPivot`	100,000 sales rows in a pivot grid with a field list, dates grouped by year, quarter and month, a sort by value, a colour scale and CSV, HTML and PNG export. The natural next step after demo 05.
`DataSetPivot`	A pivot over a `TClientDataSet` with 400 orders that follows every posted change, and a double click that lists the orders behind a cell.
`AdvancedPivot`	The features of the Registered editions: a calculated field, a calculated item, show values as, Top N with an Others row, saving and loading the layout as JSON, and the XLSX export.
`MillionRows`	One million generated rows, with Sum, CountDistinct and StdDev values and the time of every load and recalculation.
`GridToPivotToChart`	An sgcGrid feeds a pivot through `TsgcPivotSource_Table`, and the pivot result feeds an sgcCharts chart, with no code in between.
`PivotChart`	A pivot chart: field buttons above the chart, a chart designer, and chart settings that travel with the pivot layout.
`AskYourPivot`	A query bar that turns plain language questions such as “top 3 countries by sales” into pivot layouts through an AI provider.
`HeadlessReport`	A console application that reads a CSV file into a `TsgcPivotEngine` and writes PNG, SVG, HTML, CSV and JSON reports without a window, on Windows and Linux64.
`SalesPivotFMX`	The FireMonkey pivot grid and field list over 50,000 generated sales rows, with touch pan and a pivot chart tab, for Windows, macOS and Android.

Some of these demos use features or products that are not part of every edition, such as the advanced features, sgcCharts or an AI provider. Their readme files say what they need.

Help topics

The sgcPivotGrid help is part of the sgcUI help at https://www.esegece.com/help/sgcUI, in the PivotGrid section. These topics are the best continuation of this guide:

Topic	What it covers
Introduction	What sgcPivotGrid does and where to start.
Architecture	The sources, the column store, the engine, the snapshot and the views, and the threading rules.
Platforms	The Delphi versions, the target platforms and the packages.
Areas, Aggregations, Date grouping	The layout in depth.
Totals, Expand and collapse, Sorting, Filters	Subtotals and grand totals, the expanded state, member and value sorting, member, label and value filters.
Drill-down, Conditional formatting, Export	The source records behind a value, format rules, and the export formats.
Troubleshooting	More problems and their causes.
One topic per class	`TsgcVCLPivotGrid`, `TsgcVCLPivotFieldList`, `TsgcPivotEngine`, `TsgcPivotField`, `TsgcPivotSource_DataSet` and the others, with every property, its type and its default.

On the website

- https://www.esegece.com/products/sgcpivotgrid/: the product page, with the features, the editions and the downloads.

Getting help

If you are stuck, write to info@esegece.com or open a ticket in the helpdesk of your account at https://www.esegece.com/. It helps to include your Delphi version, your sgcPivotGrid version and edition, and the demo or a small project that shows the problem.

Enjoy turning your data into answers.

---

sgcPivotGrid and sgcUI are products of eSeGeCe. https://www.esegece.com