sgcDashboard

Getting Started

Put tiles, KPI cards and live numbers 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 turn it into a dashboard with sgcDashboard: a screen of rounded tiles with key figures, small trend charts, lists and charts, all in the same light or dark look. You do not need to know anything about sgcDashboard or the rest of sgcUI in advance.

The guide follows the five QuickStart demos. Each demo adds one idea to the previous one, from three KPI cards on a grid to a small sales dashboard with live numbers, a chart tile, a theme switch and a button that publishes the whole board as a web page. 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, look at the form in the designer, then read its section and try the suggested changes. Chapters 1 to 3 explain what sgcDashboard 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\Dashboard\Delphi\00.QuickStart` folder of your sgcUI installation. Each demo has its own subfolder, from `01.HelloDashboard` to `05.DashSalesApp`, with a Delphi project (`.dpr`) that you can open and run directly. All of them are VCL applications for Windows, and they need no network, no files and no API key.

The screenshots were taken on a Windows machine with Spanish regional settings. Numbers and dates in the demos follow the Windows regional settings of the machine that runs them, so on your machine `$33.851` may read `$33,851`.

Contents

1. What sgcDashboard is

The idea in one minute
Your first dashboard needs no code at all
What you need
Part of sgcUI
Coming from panels and 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

The board is a grid
Tiles ask for a cell
A tile hosts a control
The KPI card
Feeding numbers from code
Data sources and built-in tiles
One theme for the whole board
Publish, print and export
Threads
Summary: who does what

4. The QuickStart demos

4.1 Demo 01: Hello Dashboard
4.2 Demo 02: Tile Grid
4.3 Demo 03: Live Data
4.4 Demo 04: Themes
4.5 Demo 05: Sales App

5. Main components

Reference table
The dashboard and its tiles
The KPI card
The data source and the built-in tiles
Beyond the basics

6. Common problems

Layout and tiles
Themes and colours
Data, publishing and compiling

7. Where next

The other Delphi demos
Help topics
On the website
Getting help

1. What sgcDashboard is

sgcDashboard adds dashboards to Delphi applications, for the VCL and for FireMonkey. A dashboard is the screen that answers "how are we doing right now?" at a glance: a few key figures with an arrow that says whether they went up or down, a small line chart under each one, and next to them a list, a log or a chart with more detail.

You drop a `TsgcVCLDashboard` on a form, add tiles to it, and put a control into every tile: most often a `TsgcVCLKPICard`, the card with a big number and a sparkline, but just as well a `TListBox`, a `TMemo` or a chart. The dashboard places and sizes the tiles on a grid of columns, so the layout keeps its proportions when the window grows, and one `Theme` property gives every tile and every card the same light or dark look.

This guide is for Delphi developers who are new to sgcDashboard. 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

Think of a dashboard as a sheet of squared paper with twelve columns. Every tile says where it starts and how many squares it covers:

Tile	Col	Row	ColSpan	RowSpan	Result
Sales	0	0	4	2	The first third of the width, two rows high
Orders	4	0	4	2	The middle third
Visitors	8	0	4	2	The last third

Three tiles of four columns fill the twelve columns exactly. Make the window wider and every column grows, so the three tiles stay one third each. You never set the pixel position of a tile; you say which cells it covers and the dashboard does the arithmetic.

Inside each tile sits a control. A KPI card shows the four things a key figure needs:

- **A title and a value**, such as "Orders" and "212". The value is text, so you format it the way you like.
- **A trend**: an up, down or flat arrow with a short text next to it, such as "+3.9%".
- **A sparkline**: a small line chart of the last values, filled one point at a time with `AddValue`.
- **A footer** with a line of context, such as "Shipped this month".

That is the entire idea: a grid, tiles on the grid, a control in every tile. Everything else is detail.

Your first dashboard needs no code at all

The first QuickStart demo is the smallest dashboard there is. Everything you see is set in the Object Inspector and stored in the form file. This is one of its three tiles, with the KPI card inside it:

```
    object tileSales: TsgcVCLDashboardTile
      Title = 'Sales'
      ShowHeader = False
      Col = 0
      Row = 0
      ColSpan = 4
      RowSpan = 2
      object kpiSales: TsgcVCLKPICard
        Title = 'Sales this month'
        Value = '$12.4K'
        Icon = '$'
        Trend = ktUp
        TrendValue = '+6.9%'
        FooterText = 'Compared with last month'
        Align = alClient
      end
    end
```

The only code in the demo draws the sparklines. A sparkline needs a series of numbers, and the form file cannot store those, so `FormCreate` adds twelve points to every card:

```
procedure TfrmHelloDashboard.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  { AddValue appends one point to the sparkline of a card. The text of the
    card (Value, TrendValue) is set in the DFM and does not change here. }
  for i := Low(CS_SALES) to High(CS_SALES) do
  begin
    kpiSales.AddValue(CS_SALES[i]);
    kpiOrders.AddValue(CS_ORDERS[i]);
    kpiVisitors.AddValue(CS_VISITORS[i]);
  end;
end;
```

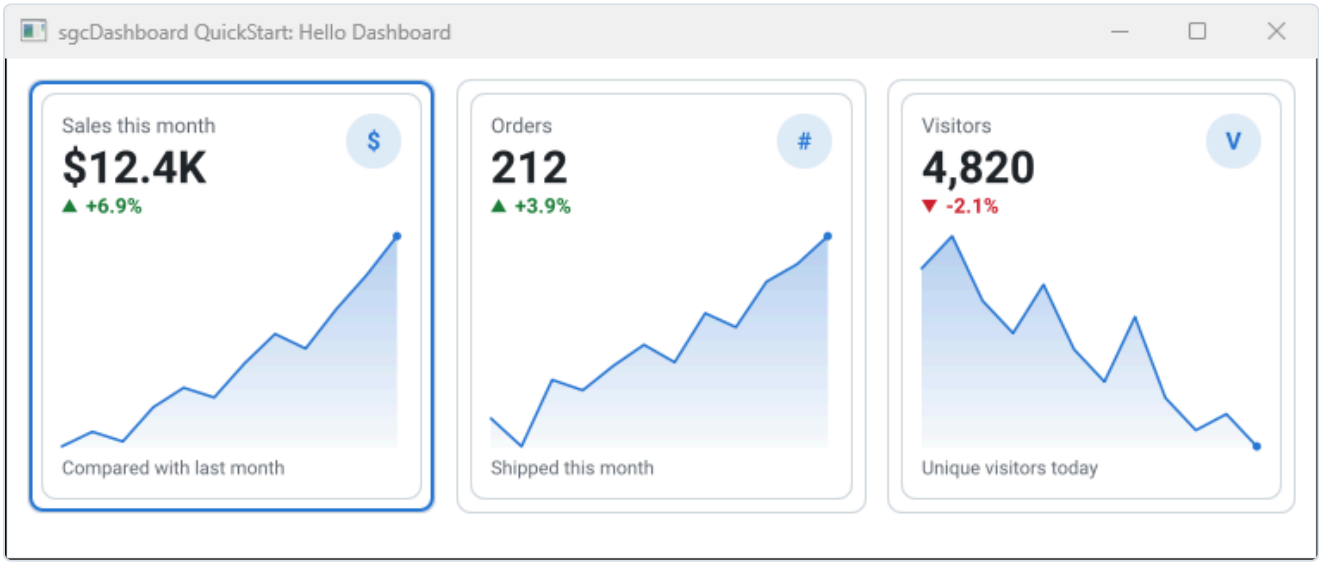


Figure 1.1. The result of demo 01: three tiles of four columns, a KPI card in each.

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.** `TsgcVCLDashboard`, `TsgcVCLDashboardTile` and `TsgcVCLKPICard` run on Windows, Win32 in every version and Win64 from Delphi XE2 on.
- **FireMonkey.** `TsgcFMXDashboard`, `TsgcFMXDashboardTile` and `TsgcFMXKPICard` have their own packages from Delphi XE6 on, with the same properties, methods and events as the VCL classes.
- **No window at all.** The layout engine and the KPI card model use no visual framework, so a KPI card can also be written to a PNG or SVG file from a console application or a server.
- **sgcUICore.** The shared foundation of the sgcUI visual products. It is installed together with sgcDashboard, you do not install it separately.
- **sgcCharts, only for chart tiles.** Demo 05 shows a chart tile, which comes from the sgcCharts bridge of sgcDashboard. Demos 01 to 04 need sgcDashboard only.

Part of sgcUI

sgcDashboard is one product of sgcUI, the eSeGeCe suite of visual components (grids, charts, pivot grids, gauges, maps 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 three things for you:

- Tiles and KPI cards look the same on the VCL, on FireMonkey and in the headless PNG and SVG output, and they match the other sgcUI controls on your form. The `Theme` property (`utDefault`, `utLight`, `utDark`, `utCustom`, `utHighContrast`) works the same way on all of them.
- A tile can host the other products. With the matching bridge package installed, a tile already contains an sgcCharts chart, an sgcGauges gauge, an sgcGrid grid, an sgcPivotGrid pivot grid or an sgcMaps map, filled from a shared data source.
- The dashboard core never uses one of those products, because sgcDashboard is also sold on its own. Each built-in tile lives in its own bridge package and appears only when its product is installed.

Coming from panels and grids

Many Delphi dashboards start as a form full of `TPanel` controls with labels on them, aligned by hand or with `TGridPanel` or `TFlowPanel`. That works, but every new figure means another panel, another set of labels and another round of sizing and colouring.

sgcDashboard gives you the pieces those forms keep rebuilding: a grid that places tiles by column and row, a card that already knows how to draw a value, a trend and a sparkline, and a theme that reaches every tile and the ordinary VCL controls inside them. Your existing controls keep working: a `TListBox` dropped into a tile is still a `TListBox`, it just gets a frame, a title bar and the colours of the theme.

2. Install and the palette

This chapter gets sgcDashboard 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

sgcDashboard 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 **sgcDashboard**. The installer shows one checkbox per product and enables the products your licence includes. sgcUICore has no checkbox: it is installed automatically with sgcDashboard.
4. To follow demo 05, select **sgcCharts** as well, if your licence includes it. When both products are selected, the installer also builds the chart tile bridge of sgcDashboard.
5. Optionally select **sgcDashboard for sgcWebSockets**, the bridge that feeds KPI cards from WebSocket, MQTT and SSE messages. It installs only when sgcWebSockets is present, and you do not need it for this guide.
6. 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 sgcDashboard units without any setting on your side.

The packages

You only need this section if you build the packages yourself. sgcDashboard 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
`sgcDashboard<V>`	Runtime	The layout engine, the KPI model, the data sources and the HTML writer. No VCL, no FireMonkey.
`sgcDashboardVCL<V>`	Runtime	`TsgcVCLDashboard`, `TsgcVCLDashboardTile`, `TsgcVCLKPICard` and the other VCL controls. Requires sgcUICore and sgcUICoreVCL.
`sgcDashboardFMX<V>`	Runtime, XE6 and later	The FireMonkey twins of the VCL controls.
`dclsgcDashboard<V>`	Design time	Registers the VCL components on the palette and adds the **Add Tile** command to the dashboard.
`dclsgcDashboardFMX<V>`	Design time, XE6 and later	Registers the FireMonkey components. Install `dclsgcDashboard` first.
`sgcDashboardChartsVCL<V>`, `dclsgcDashboardCharts<V>`	Runtime and design time	The chart tile, `TsgcVCLDashboardChartTile`. Requires sgcCharts. The gauge, grid, map and pivot tiles follow the same pattern.
`sgcDashboardWS<V>`, `dclsgcDashboardWS<V>`	Runtime and design time	The sgcWebSockets bridge with the WebSocket, MQTT and SSE sources.

Build them in dependency order: sgcUICore first, then `sgcDashboard`, then `sgcDashboardVCL`, then `dclsgcDashboard`, and the bridge packages last. Then install the design time packages 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 Dashboard**. In a VCL project it shows these components:

Component	Visual	What it does
`TsgcVCLDashboard`	yes	The board: places its tiles on a grid of columns and rows, and scrolls when they do not fit.
`TsgcVCLDashboardTile`	yes	A tile: a rounded card on the grid, with an optional title bar, that hosts other controls.
`TsgcVCLKPICard`	yes	A KPI card: title, value, icon, trend arrow, footer and sparkline.
`TsgcVCLActivityFeed`, `TsgcVCLAuditTrail`	yes	An activity feed (newest first, grouped by day) and an audit trail (a table of changes), both showing the items of an activity model.
`TsgcVCLDashboardActivityTile`, `TsgcVCLDashboardAuditTile`	yes	Tiles that already contain an activity feed or an audit trail.
`TsgcDashboardActivityModel`	no	The list of activity items those views show.
`TsgcVCLDashboardAI`	no	Routes a question typed in plain language to the tile it is about, through an AI provider.
`TsgcVCLDashboardDesignPanel`	yes	The side panel of the runtime designer. Only in builds with the advanced features, see Editions.

The bridge packages add their tiles to the same page: `TsgcVCLDashboardChartTile` with sgcCharts, `TsgcVCLDashboardGaugeTile` with sgcGauges, `TsgcVCLDashboardGridTile` with sgcGrid, `TsgcVCLDashboardMapTile` with sgcMaps and `TsgcVCLDashboardPivotTile` with sgcPivotGrid. The sgcWebSockets bridge adds `TsgcDashboardSource_WebSocket`, `TsgcDashboardSource_MQTT` and `TsgcDashboardSource_SSE`. In a FireMonkey project the page shows the `TsgcFMX...` twins of the controls instead.

The QuickStart demos use the first three components of the table, plus the chart tile in demo 05.

> **Note.** The data source of the built-in tiles, `TsgcDashboardDataSource` in unit `sgcDashboard_Data`, is not on the palette. You create it in code and assign it to the `DataSource` property of the tile, as demo 05 does.

Help at design time

Two things make the designer pleasant to work with:

- **The Add Tile command.** Right click a dashboard on the form and choose **Add Tile**. A new tile appears in the first free place of the grid, with a unique name, ready for you to drop a control into it.
- **Snapping.** Moving or resizing a tile in the designer snaps it to the grid, so `Col`, `Row`, `ColSpan` and `RowSpan` follow your mouse. You can also type the four values in the Object Inspector.

To put a control into a tile, select the control on the palette and click inside the tile, exactly as you would drop a control on a `TPanel`. Then set its `Align` to `alClient` so it fills the tile.

Editions

sgcDashboard is sold on its own, and it is also part of sgcUI Professional, sgcUI Enterprise and the All-Access bundle. The trial edition has the same features and is meant for evaluation.

A few members belong to the advanced features and are compiled only when `SGC_DASHBOARD_PERSIST` and `SGC_DASHBOARD_ADVANCED` are defined in `sgcVer.inc`: runtime drag and resize with `AllowDrag` and `AllowResize`, `SaveLayout` and `LoadLayout`, the runtime designer with `EditMode`, `AddTile` and `Undo`, and cross filtering with the filter bar. Registered and trial builds define both; a reduced Community build turns them off.

> **Note.** The QuickStart demos deliberately avoid the advanced features, so they compile and run in every build. Chapter 7 lists the `SaveLayout` and `Designer` demos, which show them.

Where the demos are

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

Folder	Project	What it shows
01.HelloDashboard	`HelloDashboard.dpr`	The smallest dashboard: three tiles with a KPI card in each
02.DashTileGrid	`DashTileGrid.dpr`	Tiles of different sizes, title bars, auto placement, ordinary VCL controls in tiles
03.DashLiveData	`DashLiveData.dpr`	A timer feeds three KPI cards every second; a click on a card lists its last values
04.DashThemes	`DashThemes.dpr`	One `Theme` property for the whole board, accent colours per card
05.DashSalesApp	`DashSalesApp.dpr`	A small sales dashboard: live cards, a chart tile, a theme switch and publishing to HTML

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\Dashboard\Delphi` are larger demos, listed in chapter 7.

Running your first demo

1. In Delphi, choose **File** > **Open Project** and open `Demos\Dashboard\Delphi\00.QuickStart\01.HelloDashboard\HelloDashboard.dpr`.
2. Press **F9**. The window opens with three KPI cards.
3. Resize the window. The three tiles keep one third of the width each.
4. Back in the IDE, open the form and click one of the tiles. Its `Col`, `Row`, `ColSpan` and `RowSpan` 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 dashboard does. Read them once now, and come back to them when a demo surprises you.

The board is a grid

A `TsgcVCLDashboard` divides its width into columns and its height into rows of a fixed height. Four properties describe the grid:

Property	Default	Meaning
`Columns`	12	The number of columns. The column width is the width of the dashboard, less the padding and the gaps, divided by `Columns`.
`RowHeight`	120	The height of one row. The dashboard has as many rows as its tiles need.
`Gap`	12	The space between two tiles, horizontally and vertically.
`Padding`	12	The space between the border of the dashboard and the grid.

All sizes are device independent pixels: on a monitor at 150% they are scaled by 1.5, so the layout looks the same at every DPI. Twelve columns is a good default because twelve divides by 2, 3, 4 and 6: two halves, three thirds, four quarters or six sixths all fit exactly. The columns follow the width of the window, the rows do not: when the tiles need more rows than fit, the dashboard scrolls vertically.

Tiles ask for a cell

Every tile asks for a cell with four properties. `Col` and `Row` are zero based and say where the tile starts; `ColSpan` and `RowSpan` say how many columns and rows it covers. A new tile has `Col` and `Row` set to -1 and a span of 3 columns by 2 rows.

The dashboard places the tiles in two passes:

1. **Explicit cells.** The tiles with both `Col` and `Row` set are placed first, in control order. When a cell overlaps a tile already placed, the tile is pushed down one row at a time until its cell is free. Its columns never change.
2. **Auto flow.** While `AutoFlow` is `True`, the default, the tiles whose `Col` or `Row` is -1 fill the first free slot, scanning the rows from the top and the columns from the left.

Before placing a tile, the dashboard clamps its cell to the grid: `ColSpan` to between 1 and `Columns`, and `Col` so the tile does not pass the last column. A hidden tile takes no cell. The properties keep what you asked for; the cell the tile really got is in `Dashboard.Layout.PlacedCell[Index]`. Demo 02 uses both passes.

A tile hosts a control

A tile is a real control inside the dashboard, much like a `TTabSheet` inside a `TPageControl`. It paints a card surface with a border, rounded corners and, when `ShowHeader` is `True` (the default), a title bar with its `Title`. The rest of the tile is client area for the controls it hosts. A KPI card shows its own title, which is why the demos set `ShowHeader = False` on the tiles that hold one.

At design time you add tiles with the **Add Tile** command. At run time you create a tile, set its cell and set its `Parent` to the dashboard; a control goes into the tile the same way. This procedure adds a tile with a KPI card in code:

```
procedure AddKPITile(aDashboard: TsgcVCLDashboard; const aTitle,
  aValue: string; aCol: Integer);
var
  oTile: TsgcVCLDashboardTile;
  oCard: TsgcVCLKPICard;
begin
  oTile := TsgcVCLDashboardTile.Create(aDashboard.Owner);
  oTile.Title := aTitle;
  oTile.ShowHeader := False;
  oTile.Col := aCol;
  oTile.Row := 0;
  oTile.ColSpan := 4;
  oTile.RowSpan := 2;
  oTile.Parent := aDashboard;

  oCard := TsgcVCLKPICard.Create(aDashboard.Owner);
  oCard.Align := alClient;
  oCard.Title := aTitle;
  oCard.Value := aValue;
  oCard.Trend := ktUp;
  oCard.TrendValue := '+2.5%';
  oCard.Parent := oTile;
end;
```

It needs `Controls`, `sgcDashboard_VCL`, `sgcDashboard_KPI_VCL` and `sgcDashboard_KPI` (for `ktUp`) in the `uses` clause.

The KPI card

A `TsgcVCLKPICard` is a thin control around a model, a `TsgcDashboardKPI` published as its `KPI` property. The model holds the data and paints the card; the control shows it on the form. The properties you set most:

Property	What it does
`Title`, `Value`, `FooterText`	The three texts of the card. `Value` is a string: you decide the format.
`Icon`	A short text, such as `'$'` or `'%'`, drawn in a circle of the accent colour. Empty draws no icon.
`Trend`, `TrendValue`	The arrow (`ktNone`, `ktUp`, `ktDown`, `ktNeutral`) and the text next to it. The default is `ktNone`, no arrow.
`AddValue`, `ClearValues`, `MaxPoints`	The sparkline. `AddValue` appends a point; once `MaxPoints` points are kept (50 by default), the oldest one is dropped.
`AccentColor`	The colour of the sparkline and the icon, as an ARGB value. 0, the default, uses the accent of the theme.

The card does not compute anything for you. It does not compare the new value with the old one to choose the arrow, and it does not format the number. Demo 03 shows the usual pattern: you calculate the change, set `Trend` and `TrendValue`, and format `Value` with `FormatFloat`. The points of the sparkline can be read back with `KPI.ValueCount` and `KPI.Values[i]`, oldest first.

Because the model needs no window, a card can also be written to an image, for example in a report or an e-mail sent by a server:

```
procedure SaveKPIImage(const aFileName: string);
var
  oKPI: TsgcDashboardKPI;
  i: Integer;
begin
  oKPI := TsgcDashboardKPI.Create;
  try
    oKPI.Title := 'Revenue';
    oKPI.Value := '$12.4K';
    oKPI.Trend := ktUp;
    oKPI.TrendValue := '+6.9%';
    oKPI.Theme := utDark;
    for i := 1 to 12 do
      oKPI.AddValue(8 + i * 0.4);
    oKPI.SaveToPNG(aFileName, 320, 180);
  finally
    oKPI.Free;
  end;
end;
```

This one uses `sgcDashboard_KPI` and `sgcUICore_Theme` (for `utDark`). `SaveToSVG` takes the same parameters.

Feeding numbers from code

A dashboard shows numbers that change, but sgcDashboard does not fetch them. Your code decides when there is a new value and hands it to the card: from a `TTimer`, after a database query, when a message arrives. One update usually touches four members:

1. `AddValue` adds the new point to the sparkline.
2. `Value` shows the new number as text.
3. `Trend` and `TrendValue` show the direction and the size of the change.

Every change repaints the card by itself; you never call `Invalidate`. Demos 03 and 05 feed their cards from a `TTimer`, which is the simplest way to see a dashboard move. If your numbers come from an sgcWebSockets client, the sgcDashboardWS bridge offers sources that read a value from every JSON message and push it into a card; chapter 7 points to the `LiveKPI` demo.

Data sources and built-in tiles

A KPI card holds a single figure. A chart, a gauge, a grid, a pivot grid or a map needs rows of data. For those, sgcDashboard has *built-in tiles*: tiles that already contain the control and fill it from a `TsgcDashboardDataSource`.

The data source is a small in-memory table. You fill `Columns` with the column names and call `AppendRow` with one value per column; `Clear` removes the rows and keeps the columns; `BeginUpdate` and `EndUpdate` bracket many changes so the tiles refresh once. Instead of your own rows it can also load a `TDataSet` with `LoadFromDataSet`, or follow one with the `DataSet` and `Live` properties.

A built-in tile never moves rows into its control itself. You set its `DataSource` and tell it which columns to use; for the chart tile, `ArgumentColumn` (one bar per distinct value), `ValueColumns` (one series per column) and `Aggregate` (`dagSum` by default, or `dagCount`, `dagAverage`, `dagMin`, `dagMax`, `dagNone`). From then on every row you append reaches the tile, which groups, aggregates and redraws. One data source can feed any number of tiles. Demo 05 shows the whole chain.

One theme for the whole board

The `Theme` property of the dashboard, of type `TsgcUITheme` from `sgcUICore_Theme`, takes one of these values:

- `utDefault`, the default: follows the application theme, set once for the whole program with `sgcUISetApplicationTheme`.
- `utLight` and `utDark`: a fixed light or dark look.
- `utHighContrast`: the high contrast look.
- `utCustom`: your own colours. The dashboard then leaves the hosted controls alone.

Every tile has `ParentTheme = True` by default and follows the dashboard. A tile that gets its own `Theme` stops following. The tile then pushes its theme into the controls it hosts through their published properties, with no compile time dependency on them. The first rule that matches wins:

1. A published `Theme` of type `TsgcUITheme` is set. This covers the KPI card.
2. Else a published `Chart` object with a `Theme` is themed, as in a chart of sgcCharts.
3. Else a published `BackgroundColor` takes the surface colour of the theme.
4. Else a published `Color` takes the surface colour, as in a `TMemo` or a `TListBox`. With rules 3 and 4 the text colour is set too, through `Font.Color`, so the text stays readable on a dark surface.

Only the direct child controls of a tile are themed. Controls outside the dashboard, such as a toolbar panel at the top of the form, are not; `sgcUIApplyThemeVCL` from `sgcUICore_Theme_VCL` themes them with the same rules, as demos 04 and 05 do. If you change hosted controls in code, call `Dashboard.ApplyTheme` to push the theme again.

Publish, print and export

A dashboard can leave the screen in three ways, all in the dashboard itself:

- **Publish to web.** `SaveToHTML`, `SaveToHTMLStream` and `AsHTML` write the dashboard as one self contained HTML page, with no script library, style sheet or image file. Every tile is drawn as inline SVG at its layout size. The `HTMLOptions` property sets the page title, a light, dark or browser following theme, and an optional refresh URL. Demo 05 publishes its board this way.
- **Print.** `Print` prints the dashboard, with the print dialog by default, and `ShowPrintPreview` shows a preview first.
- **PDF.** `ExportToPDF` writes a PDF file when the sgcUI PDF export is linked into the program.

The page and the printout draw every sgcUI control in a tile with its own renderer: KPI cards, charts, gauges, grids and pivot grids. Ordinary VCL controls, such as a `TListBox`, are left blank.

Threads

The dashboard, the tiles and the cards are VCL controls, so treat them as such: create them, change their properties and read them in the main thread. A `TTimer`, as used in the demos, runs in the main thread, so it needs no special care.

Two parts are safe to call from other threads: `AddValue` and `ClearValues` of a KPI model, and the views of a data source, which are rebuilt on the first read after the data changed. The texts of a card (`Value`, `Trend`, `TrendValue`) are ordinary properties; from a worker thread, set them through `TThread.Queue` or `TThread.Synchronize`, which is what the sgcDashboardWS sources do for you.

Summary: who does what

Piece	Its job
`TsgcVCLDashboard`	The grid: columns, row height, gaps, auto flow, scrolling, the theme, publishing and printing.
`TsgcVCLDashboardTile`	One cell of the grid: frame, title bar, and a home for any control. Passes the theme on.
`TsgcVCLKPICard`	One key figure: value, trend, icon, footer and sparkline.
`TsgcDashboardDataSource`	Rows of data shared by the built-in tiles. Created in code.
`TsgcVCLDashboardChartTile` and the other built-in tiles	A tile with a chart, gauge, grid, pivot grid or map that fills itself from a data source.
Your code	Decides when a number changes and hands it to the cards or the data source.

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.HelloDashboard	`HelloDashboard.dpr`	The dashboard, tiles on a 12 column grid, KPI cards set at design time, `AddValue`
02.DashTileGrid	`DashTileGrid.dpr`	Tiles of different sizes, title bars, `AutoFlow`, ordinary VCL controls in tiles. No code.
03.DashLiveData	`DashLiveData.dpr`	Feeding cards from a timer, trend and format in code, a shared `OnClick`, reading the sparkline values
04.DashThemes	`DashThemes.dpr`	One `Theme` for the whole board, `AccentColor`, theming a plain `TPanel`
05.DashSalesApp	`DashSalesApp.dpr`	A data source and a chart tile, live cards, publishing to HTML

> **Note.** The demos need sgcDashboard installed, so the components appear on the SGC Dashboard palette page and the `sgcUI\Source` folder is on the library path. Demo 05 also needs sgcCharts and the chart tile bridge. Random looking data uses a fixed `RandSeed := 2026`, so every run starts with the same figures as the screenshots, apart from the separators of your regional settings.

4.1 Demo 01: Hello Dashboard

Folder `01.HelloDashboard`, project `HelloDashboard.dpr`.

What you will build

The smallest dashboard there is: three tiles side by side, each holding a KPI card with a title, a value, an icon, a trend arrow, a footer and a sparkline. All the texts are set in the Object Inspector; the code only adds the points of the sparklines.

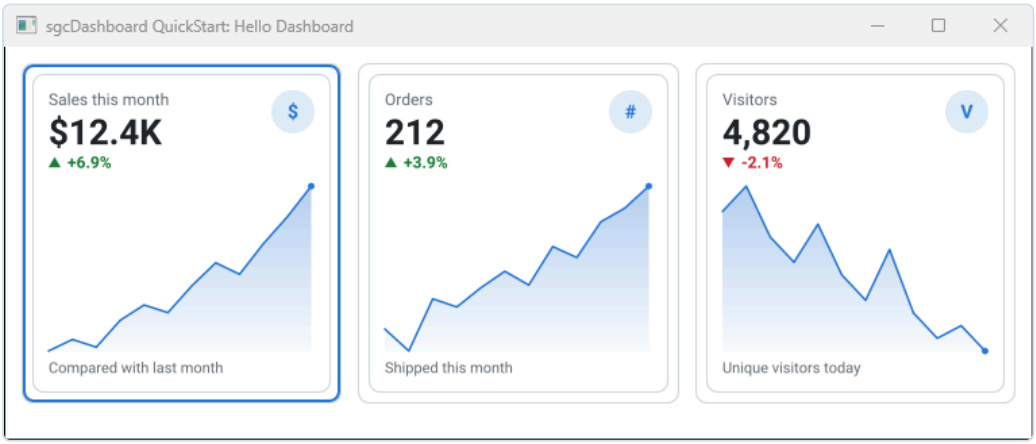


Figure 4.1. Demo 01. The blue ring around the first tile marks the keyboard tile of the focused dashboard.

Components on the form

Component	Class	Properties you set
`Dashboard`	`TsgcVCLDashboard`	`Align = alClient`. Everything else is default: 12 columns, `RowHeight = 120`.
`tileSales`, `tileOrders`, `tileVisitors`	`TsgcVCLDashboardTile`	`Col = 0`, `4` and `8`; `Row = 0`; `ColSpan = 4`; `RowSpan = 2`; `ShowHeader = False`.
`kpiSales`, `kpiOrders`, `kpiVisitors`	`TsgcVCLKPICard`	`Align = alClient`, `Title`, `Value`, `Icon`, `Trend`, `TrendValue`, `FooterText`.

Chapter 1 showed the first tile in the form file. The other two differ only in `Col` and in the texts of their card; the Visitors card has `Trend = ktDown`, which draws the red arrow.

The code

The `uses` clause names the two units of the controls on the form:

```
uses
  Classes, Controls, Forms,
  // sgc
  sgcDashboard_KPI_VCL, sgcDashboard_VCL;
```

The numbers of the sparklines are constant arrays, oldest value first:

```
const
  { the last twelve values of every metric, oldest first }
  CS_SALES: array [0 .. 11] of Double = (8.1, 8.4, 8.2, 8.9, 9.3, 9.1, 9.8,
    10.4, 10.1, 10.9, 11.6, 12.4);
  CS_ORDERS: array [0 .. 11] of Double = (160, 152, 171, 168, 175, 181, 176,
    190, 186, 199, 204, 212);
  CS_VISITORS: array [0 .. 11] of Double = (5.9, 6.1, 5.7, 5.5, 5.8, 5.4, 5.2,
    5.6, 5.1, 4.9, 5.0, 4.8);
```

And this is the only event handler of the demo:

```
procedure TfrmHelloDashboard.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  { AddValue appends one point to the sparkline of a card. The text of the
    card (Value, TrendValue) is set in the DFM and does not change here. }
  for i := Low(CS_SALES) to High(CS_SALES) do
  begin
    kpiSales.AddValue(CS_SALES[i]);
    kpiOrders.AddValue(CS_ORDERS[i]);
    kpiVisitors.AddValue(CS_VISITORS[i]);
  end;
end;
```

How it works

1. The form loads from its DFM. The dashboard reads the cells of its three tiles and places them: columns 0 to 3, 4 to 7 and 8 to 11 of the first two rows.

2. Each tile is the parent of a KPI card with `Align = alClient`, so the card fills the tile. With `ShowHeader = False` the tile draws no title bar, and the card shows its own title instead.
3. `FormCreate` adds twelve points to every card. The card scales its sparkline between the smallest and the largest point, which is why the Visitors line goes down although its numbers are close together.
4. When the window is resized, the dashboard recomputes the column width and moves the tiles. The cards repaint at the new size.

> **Tip.** The value and the sparkline are independent. `Value` is the text you want to show; `AddValue` feeds the chart. Nothing forces the last point to match the text, so keep them consistent in your own code.

Try this

- Set `ColSpan` of `tileSales` to 6, and `ColSpan` of the other two tiles to 3 with `Col = 6` and `Col = 9`. The Sales card now takes half of the width.
- Set `ShowHeader = True` on a tile. It shows its `Title` in a title bar above the card.
- Set `Dashboard.Theme` to `utDark`. The board, the tiles and the cards all turn dark.

4.2 Demo 02: Tile Grid

Folder `02.DashTileGrid`, project `DashTileGrid.dpr`.

What you will build

A dashboard with five tiles of different sizes: two KPI cards, a server list, release notes and a launch checklist. The last three are ordinary VCL controls, a `TListBox`, a `TMemo` and a `TCheckListBox`, inside tiles with a title bar. The demo has no code at all; one tile is even placed by the dashboard on its own.

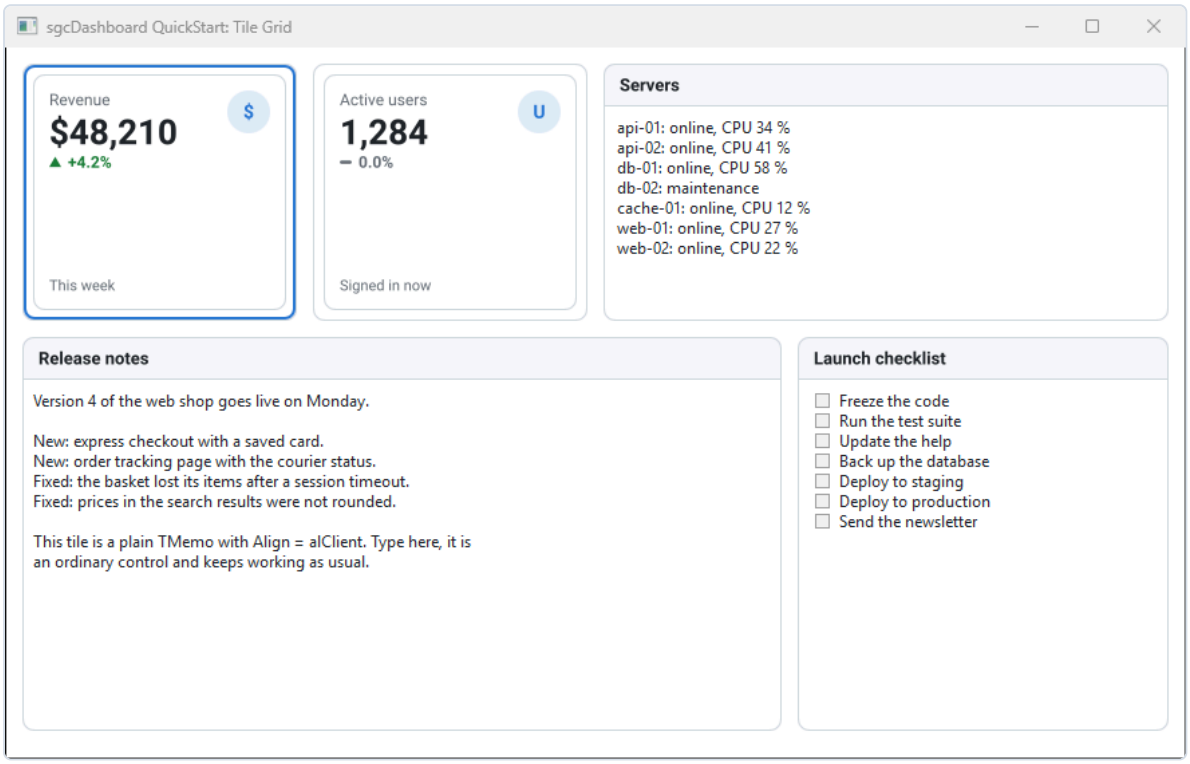


Figure 4.2. Demo 02. The Launch checklist tile has no `Col` and `Row`; auto flow put it in the first free slot.

Components on the form

Component	Class	Properties you set
`Dashboard`	`TsgcVCLDashboard`	`Align = alClient`, `RowHeight = 90`.
`tileRevenue`, `tileUsers`	`TsgcVCLDashboardTile`	Cells (0, 0) and (3, 0), 3 columns by 2 rows, `ShowHeader = False`, each with a KPI card.
`tileServers`	`TsgcVCLDashboardTile`	`Title = 'Servers'`, cell (6, 0), 6 by 2. Hosts `lstServers: TListBox`.
`tileNotes`	`TsgcVCLDashboardTile`	`Title = 'Release notes'`, cell (0, 2), 8 by 3. Hosts `memNotes: TMemo`.
`tileTasks`	`TsgcVCLDashboardTile`	`Title = 'Launch checklist'`, `ColSpan = 4`, `RowSpan = 3`, `Col` and `Row` left at -1. Hosts `chkTasks: TCheckListBox`.
`lstServers`, `memNotes`, `chkTasks`	VCL controls	`Align = alClient`, `BorderStyle = bsNone`, items or lines in the Object Inspector.

The interesting tile is the last one. Its form file has a span but no cell, because -1 is the default of `Col` and `Row` and default values are not stored:

```
  object tileTasks: TsgcVCLDashboardTile
    Title = 'Launch checklist'
    ColSpan = 4
    RowSpan = 3
    object chkTasks: TCheckListBox
      Align = alClient
      BorderStyle = bsNone
      ItemHeight = 17
      Items.Strings = (
        'Freeze the code'
        'Run the test suite'
        'Update the help'
        'Back up the database'
        'Deploy to staging'
        'Deploy to production'
        'Send the newsletter')
    end
  end
```

The code

There is none. The unit only declares the fields of the form, and the `uses` clause names the units of the controls:

```
uses
  Classes, Controls, Forms, StdCtrls, CheckLst,
  // sgc
  sgcDashboard_KPI_VCL, sgcDashboard_VCL;
```

The KPI cards of this demo therefore have no sparkline: without code there is nothing to call `AddValue`. That is intentional, and it shows that a card is useful even with only a value and a trend.

How it works

1. First pass: the four tiles with a cell are placed. Revenue and Users share the first two rows with Servers, which starts at column 6. Release notes starts at row 2 and covers columns 0 to 7.
2. Second pass: `AutoFlow` is `True`, so the Launch checklist tile, with `Col` and `Row` at -1, goes to the first free slot that holds 4 columns by 3 rows. The rows 0 and 1 are full, and in row 2 the columns 0 to 7 are taken, so it lands at column 8, row 2.
3. Each hosted control fills its tile with `Align = alClient`. The tile draws the frame, so the controls have `BorderStyle = bsNone`.
4. The tiles give the list, the memo and the checklist the colour and the font colour of the theme through their published `Color` and `Font`. The controls themselves keep working as usual: you can type in the memo and tick the tasks.

Try this

- Set `Dashboard.Columns` to 6. Spans wider than 6 are clamped to 6, and tiles that now collide are pushed down. Watch where the checklist goes.
- Set `Gap` and `Padding` to 4. The tiles move closer together and closer to the border.
- Set `RowSpan` of `tileNotes` to 5. The board becomes taller than the window and scrolls vertically.

4.3 Demo 03: Live Data

Folder `03.DashLiveData`, project `DashLiveData.dpr`.

What you will build

Three KPI cards that move: a timer produces a new revenue, order and visitor figure every second, and each card updates its value, its trend arrow and its sparkline. Click a card and the Details tile lists its last ten values; the Event log tile records what happens.

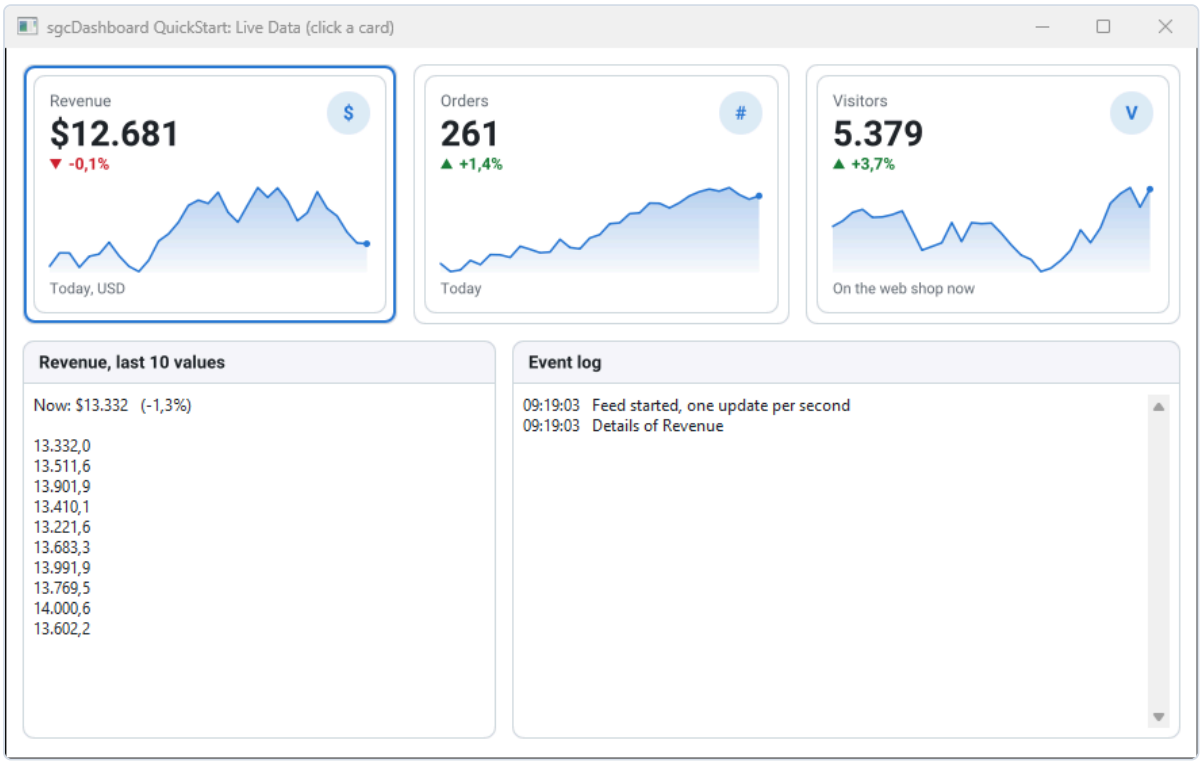


Figure 4.3. Demo 03. The Details tile is a snapshot taken at the last click (here, at start), while the cards above keep moving every second.

Components on the form

Component	Class	Properties you set
`Dashboard`	`TsgcVCLDashboard`	`Align = alClient`, `RowHeight = 90`.
`kpiRevenue`, `kpiOrders`, `kpiVisitors`	`TsgcVCLKPICard`	In tiles of 4 by 2. `Title`, `Icon`, `FooterText`, `MaxPoints = 40`, `OnClick = kpiClick` (shared). No `Value`: the code sets it.
`tileDetails`	`TsgcVCLDashboardTile`	Cell (0, 2), 5 by 3. Hosts `memDetails: TMemo` with `ReadOnly = True`.
`tileLog`	`TsgcVCLDashboardTile`	`Title = 'Event log'`, cell (5, 2), 7 by 3. Hosts `memLog: TMemo` with `ReadOnly = True`.
`tmrFeed`	`TTimer`	`Interval = 1000` (the default), `OnTimer = tmrFeedTimer`.

```
      object kpiRevenue: TsgcVCLKPICard
        Title = 'Revenue'
        Icon = '$'
        FooterText = 'Today, USD'
        MaxPoints = 40
        Align = alClient
        OnClick = kpiClick
      end
```

The code

This demo also uses `sgcDashboard_KPI`, the unit of the trend values `ktUp`, `ktDown` and `ktNeutral`:

```
uses
  SysUtils, Classes, Controls, Forms, StdCtrls, ExtCtrls,
  // sgc
  sgcDashboard_KPI, sgcDashboard_KPI_VCL, sgcDashboard_VCL;
```

`FormCreate` sets the starting figures and runs the timer handler thirty times at once, so the sparklines are full from the first second:

```
procedure TfrmDashLiveData.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  { a fixed seed, so every run starts with the same figures }
  RandSeed := 2026;
  FRevenue := 12000;
  FOrders := 180;
  FVisitors := 4800;
  { 30 ticks at once, so the sparklines are full from the first second }
  for i := 1 to 30 do
    tmrFeedTimer(nil);
  DoLog('Feed started, one update per second');
  { show the details of the first card without waiting for a click }
  kpiClick(kpiRevenue);
end;
```

`DoFeed` is the heart of the demo. It is the pattern to copy whenever a card gets a new number:

```
procedure TfrmDashLiveData.DoFeed(aCard: TsgcVCLKPICard; var aValue: Double;
  aStep: Double; const aFormat: string);
var
  vOld, vChange: Double;
begin
  { the new value: the old one plus a random step, never below zero }
  vOld := aValue;
  aValue := aValue + (Random - 0.45) * 2 * aStep;
  if aValue < 0 then
    aValue := 0;

  { 1. the sparkline }
  aCard.AddValue(aValue);
  { 2. the big text }
  aCard.Value := FormatFloat(aFormat, aValue);
  { 3. the arrow and the text next to it, the change in percent }
  vChange := 0;
  if vOld <> 0 then
    vChange := (aValue - vOld) / vOld * 100;
  if vChange > 0.05 then
    aCard.Trend := ktUp
  else if vChange < -0.05 then
    aCard.Trend := ktDown
  else
    aCard.Trend := ktNeutral;
  aCard.TrendValue := FormatFloat('+0.0;-0.0;0.0', vChange) + '%';
end;
```

The timer feeds the three cards and logs the large jumps in the orders:

```
procedure TfrmDashLiveData.tmrFeedTimer(Sender: TObject);
var
  vOld: Double;
begin
  vOld := FOrders;
  DoFeed(kpiRevenue, FRevenue, 600, '$#,##0');
  DoFeed(kpiOrders, FOrders, 12, '#,##0');
  DoFeed(kpiVisitors, FVisitors, 250, '#,##0');
  { Sender is nil for the 30 first ticks of FormCreate, log only real ones }
  if (Sender <> nil) and (Abs(FOrders - vOld) > 9) then
    DoLog(Format('Orders jumped from %.0f to %.0f', [vOld, FOrders]));
end;
```

The three cards share one click handler. It reads the sparkline back from the model of the card:

```
procedure TfrmDashLiveData.kpiClick(Sender: TObject);
var
  vCard: TsgcVCLKPICard;
  i: Integer;
begin
  { the three cards share this handler, Sender is the card clicked }
  vCard := Sender as TsgcVCLKPICard;
  tileDetails.Title := vCard.Title + ', last 10 values';
  memDetails.Lines.BeginUpdate;
  try
    memDetails.Lines.Clear;
    memDetails.Lines.Add('Now: ' + vCard.Value + '   (' + vCard.TrendValue
      + ')');
    memDetails.Lines.Add('');
    { KPI.Values holds the sparkline points, the newest is the last one }
    for i := vCard.KPI.ValueCount - 1 downto vCard.KPI.ValueCount - 10 do
      if i >= 0 then
        memDetails.Lines.Add(FormatFloat('#,##0.0', vCard.KPI.Values[i]));
  finally
    memDetails.Lines.EndUpdate;
  end;
  DoLog('Details of ' + vCard.Title);
end;
```

`DoLog` adds a line with the time to `memLog`.

How it works

1. Every second the timer calls `DoFeed` once per card. Each call adds a point, sets the text, and chooses the arrow from the change in percent, with a small dead band so tiny moves show the flat arrow.
2. Each card keeps at most 40 points (`MaxPoints = 40`). After 40 seconds the oldest point is dropped at every tick, so the sparkline scrolls to the left.
3. A click on any card calls `kpiClick` with the card as `Sender`. The handler changes the `Title` of the Details tile at run time and lists the newest ten points of `KPI.Values`, walking backwards from `KPI.ValueCount - 1`.
4. The card repaints itself after every change. The code never calls `Invalidate` or `Repaint`.

Try this

- Set `tmrFeed.Interval` to 250. The cards update four times per second.
- Set `kpiOrders.MaxPoints` to 10. Its sparkline shows only the last ten seconds.
- Add a button whose `OnClick` calls `kpiVisitors.ClearValues`. The sparkline starts again from nothing, while the value stays.

4.4 Demo 04: Themes

Folder `04.DashThemes`, project `DashThemes.dpr`.

What you will build

A dashboard that starts dark, with a combo box above it to switch to Light, High contrast or Follow Windows. One property, `Dashboard.Theme`, changes the board, the six tiles, the four KPI cards and the list and memo inside the tiles. Each card keeps its own accent colour in every theme.

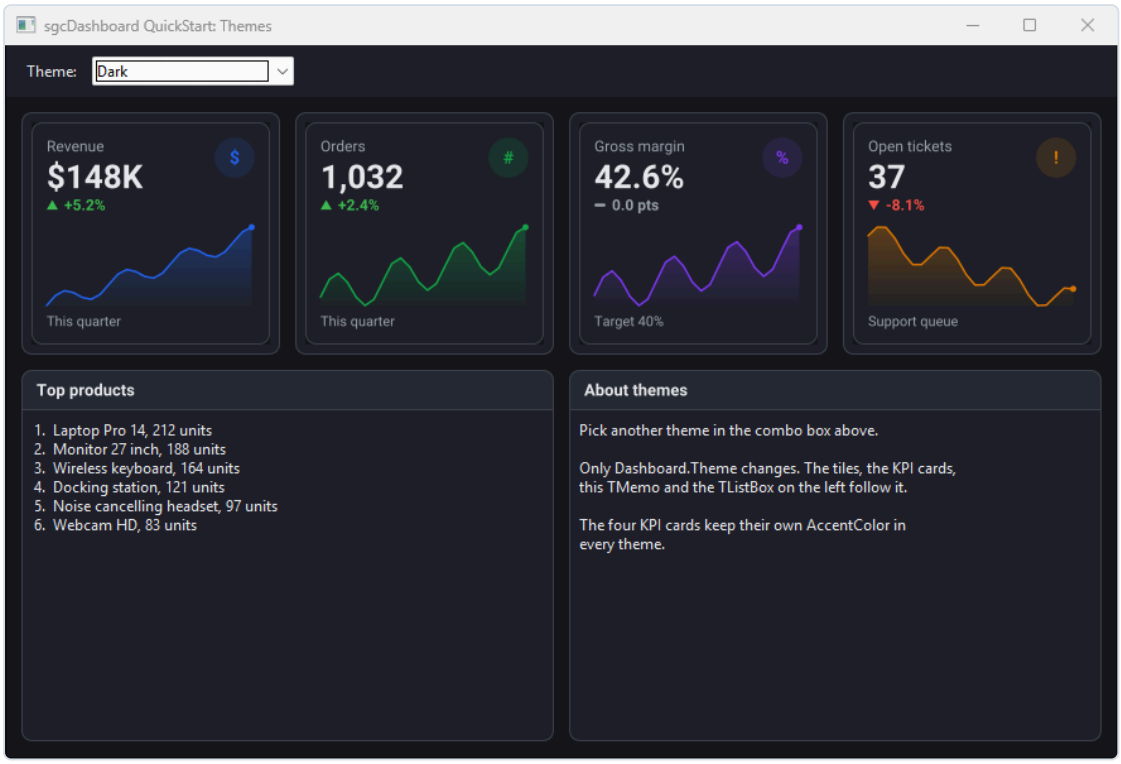


Figure 4.4. Demo 04 in the dark theme. The panel with the combo box is a plain `TPanel`, themed by one call.

Components on the form

Component	Class	Properties you set
`pnlTop`	`TPanel`	`Align = alTop`, `BevelOuter = bvNone`, `ParentBackground = False`. Holds `lblTheme` and `cbTheme`.
`cbTheme`	`TComboBox`	`Style = csDropDownList`, items Light, Dark, High contrast, Follow Windows, `ItemIndex = 1`, `OnChange = cbThemeChange`.
`Dashboard`	`TsgcVCLDashboard`	`Align = alClient`, `Theme = utDark`, `RowHeight = 90`.
`kpiRevenue`, `kpiOrders`, `kpiMargin`, `kpiTickets`	`TsgcVCLKPICard`	In tiles of 3 by 2. Texts as before, plus `AccentColor`: `$FF2563EB` blue, `$FF16A34A` green, `$FF7C3AED` purple and `$FFD97706` orange.
`tileProducts`, `tileNotes`	`TsgcVCLDashboardTile`	Cells (0, 2) and (6, 2), 6 by 3, hosting `lstProducts: TListBox` and `memNotes: TMemo`.

In the form file an accent colour looks odd. `AccentColor` is an ARGB value, and the opacity byte `$FF` sets the top bit, so the form file writes it as a negative number. `-14326805` is `$FF2563EB`:

```
      object kpiRevenue: TsgcVCLKPICard
        Title = 'Revenue'
        Value = '$148K'
        Icon = '$'
        Trend = ktUp
        TrendValue = '+5.2%'
        FooterText = 'This quarter'
        AccentColor = -14326805
        Align = alClient
      end
```

The code

Two sgcUICore units join the `uses` clause: `sgcUICore_Theme` for `TsgcUITheme` and `sgcUISystemTheme`, and `sgcUICore_Theme_VCL` for `sgcUIApplyThemeVCL`:

```
uses
  Classes, Controls, Forms, StdCtrls, ExtCtrls,
  // sgc
  sgcUICore_Theme, sgcUICore_Theme_VCL, sgcDashboard_KPI_VCL,
  sgcDashboard_VCL;
```

`DoSeed` fills a sparkline with a gentle wave, and `FormCreate` seeds the four cards:

```
procedure TfrmDashThemes.DoSeed(aCard: TsgcVCLKPICard;
  aBase, aSlope, aWave: Double);
var
  i: Integer;
begin
  { 24 points: a straight line (aBase, aSlope) with a wave on top of it }
  for i := 0 to 23 do
    aCard.AddValue(aBase + aSlope * i + aWave * Sin(i * 0.9));
end;

procedure TfrmDashThemes.FormCreate(Sender: TObject);
begin
  DoSeed(kpiRevenue, 100, 2, 6);
  DoSeed(kpiOrders, 80, 1, 9);
  DoSeed(kpiMargin, 40, 0.2, 2);
  DoSeed(kpiTickets, 60, -1, 5);
  { the combo box starts on Dark, like the Theme set in the DFM }
  cbThemeChange(nil);
end;
```

The whole theme switch is one handler:

```
procedure TfrmDashThemes.cbThemeChange(Sender: TObject);
var
  vTheme: TsgcUITheme;
begin
  case cbTheme.ItemIndex of
    0:
      vTheme := utLight;
    2:
      vTheme := utHighContrast;
    3:
      { sgcUISystemTheme reads the light or dark setting of Windows }
      vTheme := sgcUISystemTheme;
  else
    vTheme := utDark;
  end;
  { one property for the whole dashboard }
  Dashboard.Theme := vTheme;
  { the panel above the dashboard is a plain TPanel, theme it too }
  sgcUIApplyThemeVCL(pnlTop, vTheme);
end;
```

How it works

1. Setting `Dashboard.Theme` repaints the board with the new background. Every tile has `ParentTheme = True`, so every tile takes the new theme too.
2. Each tile passes the theme to the controls it hosts. The KPI cards publish a `Theme` property and get the theme itself; the list box and the memo get the surface colour in `Color` and the text colour in `Font.Color`.
3. The accent colours stay, because a non zero `AccentColor` overrides the accent of the theme. Only the surfaces and the texts change.
4. `pnlTop` is outside the dashboard, so the dashboard does not theme it. `sgcUIApplyThemeVCL` applies the same rules to the panel and, because it recurses by default, to the label and the combo box on it.
5. "Follow Windows" calls `sgcUISystemTheme`, which returns `utLight` or `utDark` from the app mode setting of Windows.

> **Watch out.** A `TPanel` shows the theme colour only with `ParentBackground = False`. With the default `True` and VCL styles or themes enabled, the panel paints the background of its parent and the call seems to do nothing. The combo box with `csDropDownList` is a native control and stays light in the dark theme; that is cosmetic only.

Try this

- Set the `AccentColor` of a card back to 0. It now uses the accent colour of the theme.
- Set `ParentTheme = False` and `Theme = utLight` on `tileProducts`. That tile stays light whatever the combo box says.
- Set `Dashboard.Theme` to `utDefault` and call `sgcUISetApplicationTheme(utDark)` at the start of `FormCreate`. The dashboard now follows the application theme, the way you would theme a whole program from one place.

4.5 Demo 05: Sales App

Folder `05.DashSalesApp`, project `DashSalesApp.dpr`. This demo needs sgcCharts and the chart tile bridge (`sgcDashboardChartsVCL` and `dclsgcDashboardCharts`).

What you will build

A small sales dashboard. A timer brings in a new order every two seconds. Four KPI cards show the revenue, the number of orders, the average order and the units sold; a chart tile shows the revenue by region as a bar chart; a list shows the latest orders. A combo box switches between light and dark, and a button publishes the whole board as an HTML page.

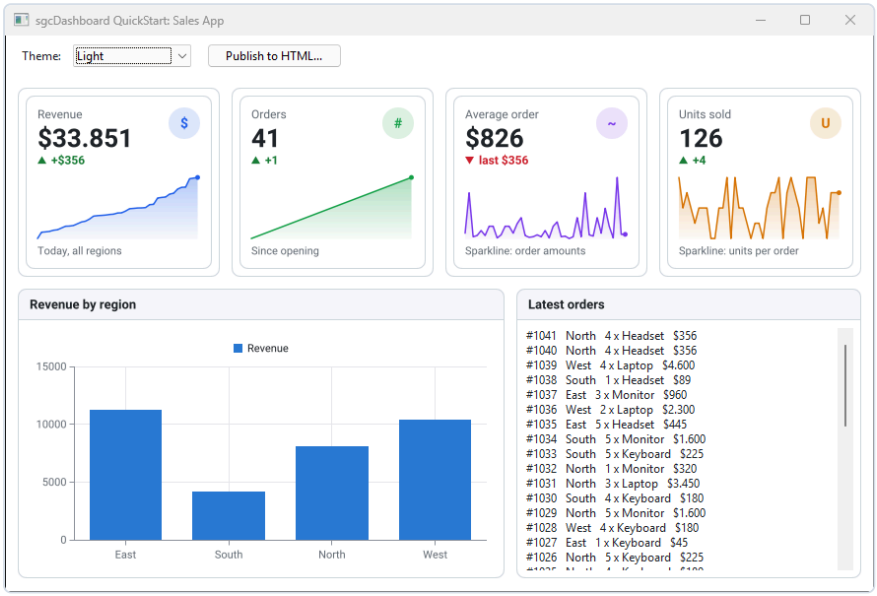


Figure 4.5. Demo 05. The bars of the chart tile grow by themselves as orders arrive; no code touches the chart.

Components on the form

Component	Class	Properties you set
`pnlTop`	`TPanel`	`Align = alTop`, `ParentBackground = False`. Holds `cbTheme` (Light, Dark) and `btnPublish`.
`Dashboard`	`TsgcVCLDashboard`	`Align = alClient`, `RowHeight = 90`, `HTMLOptions.Title = 'Sales overview'`.
`kpiRevenue`, `kpiOrders`, `kpiAverage`, `kpiUnits`	`TsgcVCLKPICard`	In tiles of 3 by 2. `Title`, `Icon`, `FooterText`, `AccentColor`.
`tileRegions`	`TsgcVCLDashboardChartTile`	`Title = 'Revenue by region'`, cell (0, 2), 7 by 3, `ArgumentColumn = 'Region'`, `ValueColumns = Revenue`, `ChartKind = tlkBar`.
`tileLatest`	`TsgcVCLDashboardTile`	`Title = 'Latest orders'`, cell (7, 2), 5 by 3. Hosts `lstOrders: TListBox`.
`tmrOrders`	`TTimer`	`Interval = 2000`, `OnTimer = tmrOrdersTimer`.
`dlgPublish`	`TSaveDialog`	`DefaultExt = 'html'`, `FileName = 'sales.html'`, a filter for HTML pages.

The chart tile is set entirely in the Object Inspector, except for its data source:

```
  object tileRegions: TsgcVCLDashboardChartTile
    Title = 'Revenue by region'
    Col = 0
    Row = 2
    ColSpan = 7
    RowSpan = 3
    ArgumentColumn = 'Region'
    ValueColumns.Strings = (
      'Revenue')
    ChartKind = tlkBar
  end
```

The code

The `uses` clause adds `sgcDashboard_Data` for the data source and `sgcDashboardCharts_Tile_VCL` for the chart tile, plus `ShellAPI` to open the published page:

```
uses
  Windows, SysUtils, Classes, Controls, Forms, StdCtrls, ExtCtrls, Dialogs,
  ShellAPI,
  // sgc
  sgcUICore_Theme, sgcUICore_Theme_VCL, sgcDashboard_Data, sgcDashboard_KPI,
  sgcDashboard_KPI_VCL, sgcDashboard_VCL, sgcDashboardCharts_Tile_VCL;
```

The form keeps the data source and a few running totals in private fields (`FSales: TsgcDashboardDataSource`, `FOrderNo`, `FRevenue`, `FUnits`). `FormCreate` builds the data source, adds forty orders and links the chart tile:

```
procedure TfrmDashSalesApp.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  { the data source: a small in-memory table. Columns names its columns,
    AppendRow adds one row with the values in the same order. }
  FSales := TsgcDashboardDataSource.Create(Self);
  FSales.Columns.Add('Region');
  FSales.Columns.Add('Product');
  FSales.Columns.Add('Units');
  FSales.Columns.Add('Revenue');

  { 40 orders to start with, so the dashboard is not empty }
  RandSeed := 2026;
  FOrderNo := 1000;
  for i := 1 to 40 do
    DoNewOrder;

  { link the chart tile to the data: from now on it follows every change }
  tileRegions.DataSource := FSales;
  cbThemeChange(nil);
end;
```

`DoNewOrder` runs for every order, at start and on every tick of the timer. It feeds the data, the cards and the list:

```
procedure TfrmDashSalesApp.DoNewOrder;
var
  vRegion, vProduct, vUnits: Integer;
  vAmount: Double;
begin
  Inc(FOrderNo);
  vRegion := Random(4);
  vProduct := Random(4);
  vUnits := 1 + Random(5);
  vAmount := vUnits * CS_PRICES[vProduct];

  { 1. the data: the chart tile updates by itself }
  FSales.AppendRow([CS_REGIONS[vRegion], CS_PRODUCTS[vProduct], vUnits,
    vAmount]);

  { 2. the KPI cards }
  FRevenue := FRevenue + vAmount;
  FUnits := FUnits + vUnits;
  kpiRevenue.AddValue(FRevenue);
  kpiRevenue.Value := FormatFloat('$#,##0', FRevenue);
  kpiRevenue.Trend := ktUp;
  kpiRevenue.TrendValue := FormatFloat('+$#,##0', vAmount);

  kpiOrders.AddValue(FOrderNo - 1000);
  kpiOrders.Value := IntToStr(FOrderNo - 1000);
  kpiOrders.Trend := ktUp;
  kpiOrders.TrendValue := '+1';
  // ...
  kpiUnits.AddValue(vUnits);
  kpiUnits.Value := IntToStr(FUnits);
  kpiUnits.Trend := ktUp;
  kpiUnits.TrendValue := '+' + IntToStr(vUnits);

  { 3. the list of the latest orders, newest on top, 50 at most }
  lstOrders.Items.Insert(0, Format('#%d   %s   %d x %s   %s',
    [FOrderNo, CS_REGIONS[vRegion], vUnits, CS_PRODUCTS[vProduct],
    FormatFloat('$#,##0', vAmount)]));
  if lstOrders.Items.Count > 50 then
    lstOrders.Items.Delete(50);
end;
```

The timer handler only calls `DoNewOrder`. The theme switch is the one from demo 04 with two choices:

```
procedure TfrmDashSalesApp.cbThemeChange(Sender: TObject);
var
  vTheme: TsgcUITheme;
begin
  if cbTheme.ItemIndex = 1 then
    vTheme := utDark
  else
    vTheme := utLight;
  Dashboard.Theme := vTheme;
  sgcUIApplyThemeVCL(pnlTop, vTheme);
end;
```

And publishing the board takes one call:

```
procedure TfrmDashSalesApp.btnPublishClick(Sender: TObject);
begin
  { SaveToHTML writes the whole dashboard as one self contained page: the
    grid, the KPI cards and the chart, with no external files. Open it in
    any browser or put it on a web server. }
  if not dlgPublish.Execute then
    Exit;
  Dashboard.SaveToHTML(dlgPublish.FileName);
  ShellExecute(Handle, 'open', PChar(dlgPublish.FileName), nil, nil,
    SW_SHOWNORMAL);
end;
```

How it works

1. `TsgcDashboardDataSource` is not on the palette, so `FormCreate` creates it with the form as owner, which also frees it. Its four columns are named in the order the values of `AppendRow` follow.
2. Forty orders are appended before the tile is linked, then `tileRegions.DataSource := FSales` connects them. The tile groups the rows by `Region`, sums `Revenue` per region (the default `Aggregate` is `dagSum`) and draws one bar per region, in the order the regions first appear.
3. Every later `AppendRow` notifies the tile, which rebuilds its view and redraws the bars. The chart is never touched in code.
4. The cards are fed exactly as in demo 03: `AddValue`, `Value`, `Trend`, `TrendValue`. The Revenue card adds the running total, so its sparkline only climbs; the Average card adds the amount of each order, so its sparkline jumps.
5. `SaveToHTML` writes the page with the title from `HTMLOptions.Title`. The tiles are drawn as inline SVG: the cards and the chart appear as on screen, while the Latest orders tile, a plain `TListBox`, stays empty in the page.

Try this

- Set `ChartKind` of `tileRegions` to `tlkPie` or `tlkDonut`. `tlkLine`, `tlkArea` and `tlkStackedBar` exist too.
- Set `ArgumentColumn` to `Product`. The chart now shows the revenue per product.
- Set `Aggregate` to `dagCount`. The bars count the orders per region instead of summing the revenue.

5. Main components

This chapter is a map of the components you met in the demos and of the ones you will meet next. The help has one topic per class with every property, its type and its default.

Reference table

Class	Palette page	What it is for
`TsgcVCLDashboard`	SGC Dashboard	The board: grid layout, scrolling, theme, publishing, printing.
`TsgcVCLDashboardTile`	SGC Dashboard	A tile that hosts any control, with an optional title bar.
`TsgcVCLKPICard`	SGC Dashboard	A key figure with trend, icon, footer and sparkline.
`TsgcDashboardDataSource`	none, create in code	Rows of data shared by the built-in tiles.
`TsgcVCLDashboardChartTile`	SGC Dashboard, with sgcCharts	A tile with a chart fed by a data source.
`TsgcVCLDashboardGaugeTile`	SGC Dashboard, with sgcGauges	A tile with a gauge showing one aggregated value.
`TsgcVCLDashboardGridTile`	SGC Dashboard, with sgcGrid	A tile with a grid of the rows of a data source.
`TsgcVCLDashboardPivotTile`	SGC Dashboard, with sgcPivotGrid	A tile with a pivot grid over the rows of a data source.
`TsgcVCLDashboardMapTile`	SGC Dashboard, with sgcMaps	A tile with a map, one marker per row with a latitude and a longitude.
`TsgcDashboardActivityModel`	SGC Dashboard	A list of activity items: who did what, and when.
`TsgcVCLActivityFeed`, `TsgcVCLAuditTrail`	SGC Dashboard	Views of an activity model, as a feed or as an audit table.
`TsgcVCLDashboardActivityTile`, `TsgcVCLDashboardAuditTile`	SGC Dashboard	Tiles that contain those views.
`TsgcVCLDashboardAI`	SGC Dashboard	Routes plain language questions to the right tile.
`TsgcVCLDashboardDesignPanel`	SGC Dashboard, advanced builds	The side panel of the runtime designer.
`TsgcDashboardSource_WebSocket`, `_MQTT`, `_SSE`	SGC Dashboard, with sgcWebSockets	Push values from WebSocket, MQTT or SSE messages into KPI cards.

Every visual class has a FireMonkey twin with the same members: `TsgcFMXDashboard`, `TsgcFMXDashboardTile`, `TsgcFMXKPICard` and so on.

The dashboard and its tiles

`TsgcVCLDashboard` owns the layout engine, published read only as `Layout`, and exposes its settings directly: `Columns`, `RowHeight`, `Gap`, `Padding` and `AutoFlow`. It also carries the `Theme`, the `HTMLOptions` of the published page and the output methods `SaveToHTML`, `Print`, `ShowPrintPreview` and `ExportToPDF`.

`TsgcVCLDashboardTile` has the cell (`Col`, `Row`, `ColSpan`, `RowSpan`), the title bar (`Title`, `ShowHeader`) and the theme (`ParentTheme`, `Theme`). The built-in tiles descend from it, so everything you learned about tiles applies to them as well.

The KPI card

`TsgcVCLKPICard` publishes the texts (`Title`, `Value`, `Icon`, `TrendValue`, `FooterText`), the arrow (`Trend`), the colours (`AccentColor`, `Theme`) and the length of the sparkline (`MaxPoints`). `AddValue` and `ClearValues` manage the sparkline, and the `KPI` property gives access to the model, a `TsgcDashboardKPI`, with `ValueCount`, `Values` and the headless `SaveToPNG` and `SaveToSVG`. The usual mouse events, such as `OnClick` and `OnDblClick`, are published too.

The data source and the built-in tiles

`TsgcDashboardDataSource`, in unit `sgcDashboard_Data`, holds rows: your own (`Columns`, `AppendRow`, `Clear`, `BeginUpdate`, `EndUpdate`), a copy of a dataset (`LoadFromDataSet`), a dataset it follows (`DataSet` with `Live = True`), or the rows of a bound table (`Table`), such as an sgcGrid grid.

Each built-in tile reads the data source through a view that groups and aggregates the rows for it. The chart tile is configured with `ArgumentColumn`, `ValueColumns`, `Aggregate` and `ChartKind`; the other tiles have the properties their content needs. Every built-in tile lives in its own bridge package, which requires sgcDashboard and the product of the tile.

Beyond the basics

- **Layouts the user changes.** `AllowDrag` and `AllowResize` let the user move and resize the tiles at run time, and `SaveLayout` and `LoadLayout` store the arrangement as JSON. Advanced builds only.
- **The runtime designer.** `EditMode`, `AddTile`, `RemoveTile`, `Undo` and `Redo`, with `TsgcVCLDashboardDesignPanel`, let the user build a dashboard. Advanced builds only.
- **Cross filtering.** A click on a bar, a row or a marker filters every other tile of the same data source, and the filter bar shows the conditions. Advanced builds only.
- **Responsive layout.** With `Breakpoints.Enabled`, a narrow dashboard switches to fewer columns and to a compact or medium cell for each tile.
- **Live sources.** The sgcDashboardWS bridge feeds KPI cards from WebSocket, MQTT and SSE messages.
- **Activity.** An activity model with an activity feed and an audit trail, as controls or as tiles.
- **Ask the dashboard.** `TsgcVCLDashboardAI` sends a typed question to the tile it is about, through an AI provider.

6. Common problems

These are the problems that come up most often in a first dashboard. Each entry says what you see, why it happens and how to fix it.

Layout and tiles

A tile is not where I put it

What you see. A tile appears lower than its `Row` says, or narrower than its `ColSpan`.

Why. The dashboard never lets two tiles overlap. A tile whose cell collides with a tile placed before it is pushed down one row at a time until its cell is free. A span wider than `Columns` is clamped, and `Col` is moved so the tile does not pass the last column. The properties keep the values you typed.

Fix. Check the cells of the neighbouring tiles, or read the cell the tile really got from `Dashboard.Layout.PlacedCell[Index]`. After changing `Columns`, review the spans.

The control does not fill the tile

What you see. A list, a memo or a KPI card sits in a corner of its tile, or shows a second border inside the frame of the tile.

Fix. Set `Align = alClient` on the hosted control, and `BorderStyle = bsNone` on controls that draw their own border, as demo 02 does. The tile already draws the frame.

The tiles are too tall or the board scrolls

What you see. On a small form only the first row of tiles is visible, and the rest needs scrolling.

Why. `RowHeight` is 120 by default, so a tile with `RowSpan = 2` is about 250 pixels high. The rows do not shrink with the window; only the columns follow the width.

Fix. Lower `RowHeight`, as demos 02 to 05 do with 90, or reduce the `RowSpan` of the tiles.

A KPI card has no sparkline

What you see. The card shows its value and trend, but the area of the line chart is empty.

Why. The points of the sparkline are not stored in the form file. Without `AddValue` calls there is nothing to draw, which is what demo 02 shows.

Fix. Call `AddValue` at start, as demo 01 does, and whenever a new value arrives, as demo 03 does.

A falling figure is shown in red although lower is better

What you see. Open tickets or costs go down, which is good news, and the card shows a red arrow.

Why. The card paints `ktDown` in red and `ktUp` in green. It has no “lower is better” option.

Fix. Choose the trend from the meaning rather than from the sign, for example `ktNeutral` with a `TrendValue` such as “8 fewer”, or show the change of a figure where up is good.

Themes and colours

A TPanel does not take the theme colour

What you see. `sgcUIApplyThemeVCL(pnlTop, utDark)` runs, and the panel stays light.

Why. With `ParentBackground = True`, the default, and themes enabled, the panel paints the background of its parent and ignores its `Color`.

Fix. Set `ParentBackground = False` on the panel, as demos 04 and 05 do.

A hosted control keeps its colours when the theme changes

What you see. Most tiles turn dark, but one control in a tile stays light.

Why. The theme reaches a hosted control only through a published `Theme`, `Chart.Theme`, `BackgroundColor` or `Color`. Only the direct child controls of a tile are themed, so a control inside a panel inside the tile keeps its colours. With `utCustom` the hosted controls are left alone on purpose. Native controls such as a `TComboBox` with `csDropDownList` draw themselves and stay light.

Fix. Put the control directly into the tile, or theme the nested controls yourself with `sgcUIApplyThemeVCL`. After changing hosted controls in code, call `Dashboard.ApplyTheme`.

An accent colour does not show

What you see. You set `AccentColor` and the sparkline and the icon disappear or keep the theme colour.

Why. sgcUI colours are 32 bit ARGB values, `$AARRGGBB`. The first byte is the opacity, and 0 means fully transparent, so `$2563EB` has no opacity. A VCL `TColor` such as `clBlue` stores its bytes in a different order and is not an ARGB value. An `AccentColor` of 0 means "use the theme accent".

Fix. Write the opacity byte: `$FF2563EB`. The form file shows such values as negative numbers, as in demo 04; that is expected.

Data, publishing and compiling

The DataSource of a chart tile cannot be picked in the Object Inspector

What you see. The drop down list of `DataSource` is empty, and there is no data source component on the palette.

Why. `TsgcDashboardDataSource` is not registered on the palette.

Fix. Create it in code and assign it, as demo 05 does: `FSales := TsgcDashboardDataSource.Create(Self)`, fill `Columns`, then `tileRegions.DataSource := FSales`.

The chart tile stays empty

What you see. The tile has a title bar but no bars.

Why. The tile has no `DataSource`, the data source has no rows yet, or `ArgumentColumn` and `ValueColumns` name columns that are not in `Columns`.

Fix. Assign the data source, check the column names against the ones added to `Columns`, and pass numbers, not formatted strings, for the value columns in `AppendRow`.

The chart tile is missing, or sgcDashboardCharts_Tile_VCL is not found

What you see. Demo 05 does not compile, or the chart tile is not on the palette.

Why. The chart tile lives in the bridge packages `sgcDashboardChartsVCL` and `dclsgcDashboardCharts`, which require sgcCharts. The installer builds them only when sgcDashboard and sgcCharts are both installed.

Fix. Install sgcCharts together with sgcDashboard, or build the two bridge packages yourself after the sgcCharts packages. Demos 01 to 04 do not need them.

A tile is empty in the published HTML page

What you see. In the page written by `SaveToHTML` the KPI cards and the chart look right, but the Latest orders tile of demo 05 is blank.

Why. The page draws every sgcUI control in a tile with its own renderer. Ordinary VCL controls, such as a `TListBox` or a `TMemo`, have no such renderer and are left blank. Printing follows the same rule.

Fix. For content that must appear in the page, use an sgcUI control or a built-in tile, for example a grid tile instead of a list box.

The numbers use the wrong separators

What you see. `FormatFloat('$#,##0', ...)` shows `$33.851` on one machine and `$33,851` on another.

Why. `FormatFloat` uses the thousands and decimal separators of 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, pass a `TFormatSettings` to `FormatFloat`.

Undeclared identifier SaveLayout, AllowDrag or EditMode

What you see. Code copied from a larger demo or from the help fails with E2003 Undeclared identifier, or a form raises “Property AllowDrag does not exist” when it loads.

Why. Those members are compiled only when `SGC_DASHBOARD_PERSIST` and `SGC_DASHBOARD_ADVANCED` are defined, and a Community build turns them off. A form saved with `AllowDrag` or `AllowResize` set cannot be read by such a build either.

Fix. Use a Registered or trial build, or remove those members. The QuickStart demos use none of them.

A card fed by a live source never changes in a console application

What you see. With the sgcDashboardWS bridge, the texts of a card stay the same in a console application or a service, although messages arrive.

Why. The sources apply the card texts in the main thread through `TThread.Queue`. A console application or a service has no message loop that runs the queue.

Fix. Call `CheckSynchronize` regularly in the main thread of such programs.

7. Where next

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

The other Delphi demos

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

Demo	What it shows
`Basic`	A dashboard with six tiles: four live KPI cards, an event log and a server list, with a light and dark theme switch. The natural next step after demo 03.
`SalesExplorer`	One data source with sales rows feeds six built-in tiles. Click a region bar, a grid cell, a pivot row or a map marker and every other tile follows the filter. Make the window narrow and the tiles reflow. The natural next step after demo 05.
`SaveLayout`	A dashboard the user rearranges at run time, with the layout saved to and loaded from a JSON file.
`Designer`	The runtime designer: the user adds built-in tiles from a palette, moves and resizes them, edits their properties, removes them, undoes and redoes, and saves the result as JSON.
`LiveKPI`	Three KPI cards bound to a WebSocket feed with `TsgcDashboardSource_WebSocket`, served by a `TsgcWebSocketServer` running inside the demo.
`Suite`	One dashboard whose tiles host the other eSeGeCe VCL products, all following the same light or dark theme, with a query bar for questions in plain language.

Some of these demos use features or products that are not part of every build: `SaveLayout` and `Designer` use the advanced features, `SalesExplorer` uses several built-in tiles and their products, `LiveKPI` needs sgcWebSockets, and `Suite` needs the sources of sgcCharts, sgcMaps, sgcPDF and sgcReports. Their readme files say what they need.

Help topics

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

Topic	What it covers
Introduction	What sgcDashboard does and where to start.
Platforms, Editions	The Delphi versions, the target platforms, the packages, and what each edition includes.
Layout, Responsive layout	The grid, the placement and collision rules, and the breakpoints.
Themes	The theme of the dashboard and what the theme applier sets on hosted controls.
Data sources, Built-in tiles	Rows, views and aggregates, and the chart, gauge, grid, pivot and map tiles.
Live data, Headless KPI	Feeding cards from sgcWebSockets, and KPI cards as PNG and SVG without a window.
Persistence, Runtime designer, Cross filtering	The advanced features.
Printing	Printing, the preview and the PDF export.
TsgcVCLDashboard, TsgcVCLDashboardTile, TsgcVCLKPICard	One topic per class, with every property, its type and its default.

On the website

- https://www.esegece.com/products/sgcdashboard/: 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 sgcDashboard version and edition, and the demo or a small project that shows the problem.

Enjoy building your dashboards.

---

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