
sgcCharts

Getting Started

Put a live, interactive chart 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 show their data as a chart with sgcCharts. You do not need any charting experience. If you can drop a component on a form and write an `OnCreate` handler, you can follow every step.

The guide follows the five QuickStart demos. Each demo adds one idea to the previous one, from a single line with fourteen points to a small sales application with a database grid, a live bound chart, themes and a PNG 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, move the mouse over the chart, zoom with the wheel, then read its section and try the suggested changes. Chapters 1 to 3 explain what sgcCharts 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\Charts\Delphi\00.QuickStart` folder of your sgcUI installation. Each demo has its own subfolder, from `01.HelloChart` to `05.ChartSalesApp`, with a Delphi project (`.dpr`) that you can open and run directly. All of them are VCL applications for Windows, and every one of them creates its own data in code, so none needs a database server or a network connection.

The screenshots were taken on a Windows machine with Spanish regional settings, which is why the month names in the database grid of demo 05 read "ene." and "feb.". Numbers and dates in standard VCL controls follow the Windows regional settings of the machine that runs the demo, so on your machine they will look like your own locale. The chart axis labels do not: they use a locale invariant format, as chapter 3 explains.

Contents

1. What sgcCharts is

The idea in one minute
Your first chart needs one loop
What you need
Part of sgcUI
Coming from other chart components

2. Install and the palette

Installing
The packages
The palette page
Editions
Where the demos are
Running your first demo

3. Core concepts

Model, view and series
Axes
Adding points
When the chart repaints
Buffers, capacity and the time window
Data sources
Threads
Mouse, events and the hit record
Themes and colours
Summary: who does what

4. The QuickStart demos

4.1 Demo 01: Hello Chart
4.2 Demo 02: Chart Setup
4.3 Demo 03: Chart Live Feed
4.4 Demo 04: Chart Events
4.5 Demo 05: Chart Sales App

5. Main components

Reference table
The chart and its views
The series
The data sources
Beyond the basics

6. Common problems

Compiling
Data and axes
Live data and interaction

7. Where next

The other Delphi demos
Help topics
On the website
Getting help

1. What sgcCharts is

sgcCharts adds charts to Delphi applications, for the VCL and for FireMonkey. Line, area, bar, scatter, pie, radar, heatmap, treemap, candlestick and more: you pick a series type, give it points, and the chart draws the axes, the grid, the legend and the tooltip for you. It is built for data that changes. A series can take thousands of new points per second, keep only the latest ones, and the chart repaints at a steady frame rate without blocking your form.

You drop a few components on a form, set the titles and the axes in the Object Inspector and add the points, either from your own code or from a `TDataSet`, a JSON document, a CSV file or a live feed. The user can zoom with the mouse wheel, drag a box to zoom into it, pan, and read every value in a tooltip, all without a line of code from you.

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

Every sgcCharts chart is made of three kinds of components, and once you see them the rest of the product makes sense:

Kind	Class	What it does
The chart	<code>TsgcChart</code>	A non visual component, the model. It holds the title, the subtitle, the legend, the axes, the theme, the palette and the mouse settings. It shows nothing on its own.
The view	<code>TsgcVCLChart</code>	The control you see on the form. Its <code>chart</code> property points to a <code>TsgcChart</code> , and it paints that chart, handles the mouse and exports the picture.
The series	<code>TsgcChartSeries_Line</code> , <code>TsgcChartSeries_Bar</code> , ...	One set of values drawn one way: a line, a row of bars, a pie. A series joins a chart through its own <code>Chart</code> property, and it keeps its points.

A fourth kind is optional: the *data sources*, such as `TsgcChartDataSource_DataSet`. A data source fills a series from somewhere else, so you do not have to write the loop yourself. Demo 05 uses one.

That is the entire idea. The model is separate from the view, which is why the same `TsgcChart` can be shown by a VCL control, by a FireMonkey control, or rendered to a PNG file in a console program with no window at all.

Your first chart needs one loop

The first QuickStart demo is the smallest chart there is: a line with the number of website visitors of fourteen days. The three components are set at design time. Seen as text, the form file holds this (the positions and sizes are left out):

```

object VCLChart: TsgcVCLChart
    Chart = Chart
    Align = alClient
end
object Chart: TsgcChart
    Title.Text = 'Website visitors per day'
    Legend.Visible = False
end
object serVisits: TsgcChartSeries_Line
    Chart = Chart
    Title = 'Visitors'
end

```

The only code adds the points:

```

procedure TfrmHelloChart.FormCreate(Sender: TObject);
var
    i: Integer;
begin
    { one AddXY per point: X is the day (1 to 14), Y the visitors. The chart
      repaints by itself, there is no Refresh to call. }
    for i := Low(CS_VISITS) to High(CS_VISITS) do
        serVisits.AddXY(i + 1, CS_VISITS[i]);
    end;
end;

```



Figure 1.1. The result of demo 01: one line series, two automatic axes and a title.

Notice what you did not have to do. You did not create any axes: a new `TsgcChart` already has an `X` axis at the bottom and a `Y` axis on the left, and a series uses them unless you tell it otherwise. You did not set a range: the axes adjust to the data. And you did not call `Refresh` or `Invalidate`: adding a point tells the view to repaint.

What you need

- **Delphi 7 or later**, up to Delphi 13. The same source and the same component names work in every version; the packages exist for Delphi 7, 2007, 2009, 2010, XE to XE8, 10 to 10.4, 11, 12 and 13.
- **VCL**: `TsgcVCLChart` on Windows, Win32 in every version and Win64 from XE2.
- **FireMonkey**: `TsgcFMXChart` from Delphi XE6, on Windows, macOS, iOS and Android.
- **No window at all**: `TsgcChartRenderer` renders a chart to PNG or SVG in a console program or a service, also on Linux64 from Delphi 10.2.

The QuickStart demos are VCL applications and need nothing else: no database server, no DLL and no network. Demo 05 uses a `TClientDataSet` and links `MidasLib`, so it does not need `midas.dll` either.

Part of sgcUI

sgcCharts is one of the products of the sgcUI suite. The drawing itself, the canvas, the fonts, the SVG output and the themes, comes from a shared package called `sgcUICore`, which is installed automatically with sgcCharts and shared with the other sgcUI products. That is why a chart and, for example, an `sgcGrid` on the same form follow the same light, dark or high contrast theme, and why the theme type `TsgcUITheme` lives in the unit `sgcUICore_Theme` rather than in an sgcCharts unit.

Other sgcUI products build on sgcCharts: `sgcDashboard` can host charts in its tiles, `sgcReports` can print them, and `sgcPivotGrid` can feed a chart from a pivot table. The optional `sgcChartsWS` package connects charts to live `sgcWebSockets` feeds (WebSocket, MQTT, Server-Sent Events and the Binance market data). None of this is needed for the QuickStart demos.

Coming from other chart components

Most Delphi developers have met a chart component before, often the TeeChart edition that ships with the IDE. Much of what you know carries over: series, axes, a legend, a title. Three things in sgcCharts are different and worth knowing from the start:

- **The chart and the control are two components.** `TsgcChart` holds the model and `TsgcVCLChart` shows it. You set titles, axes and events on the chart, and export or print through the control.
- **Series are components of their own.** They sit on the form next to the chart, are named in the form class like any other component, and join a chart through their `Chart` property.
- **Every series stores its points in a ring buffer.** A series has a `Capacity`, and when it is full the oldest point is dropped. Combined with the chart's `TimeWindow`, that makes a scrolling real time chart a matter of two properties, as demo 03 shows.

2. Install and the palette

This chapter gets sgcCharts onto your Tool Palette and the first demo running.

Installing

sgcCharts is installed by the sgcUI installer. Run it, sign in with your license when it asks, and keep the **sgcCharts** component checked (it is listed under the products together with the other sgcUI products you own). The installer copies the source, the compiled packages and the demos, installs sgcUICore with it, and registers the design packages in every Delphi version it finds on the machine.

When sgcWebSockets is also installed, the installer offers **sgcCharts for sgcWebSockets** as well, the bridge package with the live data sources. You can leave it out for now.

When the installer finishes, start Delphi, create a new VCL application and look for the **SGC Charts** page on the Tool Palette. If it is there, you are ready.

The packages

You do not need to install packages by hand when you use the installer, but it helps to know what they are, because their names show up in the project options and in error messages. Every package name ends with the IDE suffix, for example `D13` for Delphi 13 or `D7` for Delphi 7.

Package	Kind	Contents
<code>sgcChartsD13</code>	Runtime, framework neutral	The chart model, the series, the axes, the data sources and the headless renderer. Requires <code>rt1</code> , <code>dbrt1</code> and <code>sgcUICoreD13</code> .
<code>sgcChartsVCLD13</code>	Runtime, VCL	<code>TsgcVCLChart</code> .
<code>sgcChartsFMXD13</code>	Runtime, FireMonkey (XE6 and later)	<code>TsgcFMXChart</code> .
<code>dc1sgcChartsD13</code>	Design time, VCL	Registers the components on the SGC Charts page.
<code>dc1sgcChartsFMXD13</code>	Design time, FireMonkey	Registers <code>TsgcFMXChart</code> . Requires <code>dc1sgcChartsD13</code> .
<code>sgcChartsWSD13</code> , <code>dc1sgcChartsWSD13</code>	Runtime and design time, optional	The sgcWebSockets bridge. Requires sgcWebSockets.

The split exists so that a FireMonkey application for Android or macOS never links the VCL, and a console program that renders PNG files links neither.

The palette page

All sgcCharts components are on one palette page, **SGC Charts**. In a VCL project you see:

Group	Components
Chart and views	<code>TsgcChart</code> , <code>TsgcVCLChart</code> , <code>TsgcChartRenderer</code>
Line family	<code>TsgcChartSeries_Line</code> , <code>_FastLine</code> , <code>_Area</code> , <code>_StepLine</code>
Bars and points	<code>TsgcChartSeries_Bar</code> , <code>_Scatter</code> , <code>_Bubble</code>
Financial	<code>TsgcChartSeries_Candlestick</code> , <code>_OHLC</code>
Categories	<code>TsgcChartSeries_Pie</code> , <code>_Radar</code> , <code>_Funnel</code> , <code>_Treemap</code> , <code>_Heatmap</code>
Business	<code>TsgcChartSeries_BoxPlot</code> , <code>_Waterfall</code> , <code>_RangeArea</code>
Data sources	<code>TsgcChartDataSource_Manual</code> , <code>_DataSet</code> , <code>_JSON</code> , <code>_CSV</code> , <code>_Table</code> , <code>TsgcChartTableLink</code>
Financial pack	Indicator series (<code>TsgcChartSeries_SMA</code> , <code>_EMA</code> , <code>_WMA</code> , <code>_Bollinger</code> , <code>_RSI</code> , <code>_MACD</code> , <code>_Stochastic</code> , <code>_ATR</code> , <code>_VWAP</code>), <code>TsgcChartSeries_Volume</code> and <code>TsgcChartAggregator</code>

In a FireMonkey project the page shows `TsgcFMXChart` instead of `TsgcVCLChart` ; the chart, the series and the data sources are the same components in both frameworks.

Tip. Drop the `TsgcChart` first, then the `TsgcVCLChart` , and set the control's `Chart` property right away. Then drop each series and set its `Chart` property too. The designer paints the chart as soon as the links are in place.

Editions

sgcCharts is sold on its own as a single product, and it is included in sgcUI Professional, sgcUI Enterprise and the All-Access bundle. Every Registered edition includes the whole product, the financial pack and the chart designer included. The trial has every feature unlocked for the trial period; after it ends, an application built with the trial raises an exception at start up.

The five QuickStart demos use only the core components, the chart, the view, the line, area and bar series and the dataset source.

Where the demos are

The demos are in the `Demos\Charts\Delphi` folder of your sgcUI installation:

- `00.QuickStart` holds the five demos of this guide, `01.HelloChart` to `05.ChartSalesApp` , and a `readme.html` that lists them.
- The other folders, such as `LiveLine` , `PieDonut` , `Candlestick` or `Headless` , are larger demos, one feature area each. Chapter 7 lists them.

Every demo folder has its own `readme.html` with what it shows and how to run it.

Running your first demo

1. In Delphi, choose **File, Open Project** and open
`Demos\Charts\Delphi\00.QuickStart\01.HelloChart\HelloChart.dpr` .
2. Press **F9**. The window of figure 1.1 appears.
3. Move the mouse over the line. A crosshair follows it and a tooltip shows the value of the point under the mouse.
4. Turn the mouse wheel over the chart to zoom in, drag with the right mouse button to pan, and double click to go back to the full view.
5. Close the window, open `uHelloChart.dfm` in the form designer and click each of the three components. Their properties in the Object Inspector are the whole configuration of the demo.

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

This chapter explains the ideas that the five demos rely on. You can read it before the demos or come back to it when a demo makes you curious. Each idea is short, and together they cover almost everything a first chart needs.

Model, view and series

Chapter 1 introduced the three kinds of components. Here is how they connect:

- The view, `TsgcVCLChart`, has a `Chart` property. Point it to a `TsgcChart` and the view registers itself as a listener of that chart. From then on, every change of the chart makes the view repaint.
- A series has a `Chart` property too. Setting it is what adds the series to the chart; clearing it removes the series. The order in which series join is their `Index`, which also decides their default colour and the order in the legend.
- The chart knows its series (`SeriesCount` and `Series[i]`) but does not own them. They are components on your form, freed with the form.

Everything can be done in code as well. This procedure builds a complete chart at run time, the same three components that demo 01 sets at design time:

```
procedure TForm1.BuildChart;
var
  oView: TsgcVCLChart;
  oSeries: TsgcChartSeries_Bar;
begin
  FChart := TsgcChart.Create(Self);
  FChart.Title.Text := 'Created in code';
  FChart.Legend.Visible := False;

  oView := TsgcVCLChart.Create(Self);
  oView.Parent := Self;
  oView.Align := alClient;
  oView.Chart := FChart;

  oSeries := TsgcChartSeries_Bar.Create(Self);
  oSeries.Title := 'Orders';
  oSeries.Chart := FChart;
  oSeries.AddY(12);
  oSeries.AddY(19);
  oSeries.AddY(7);
end;
```

The units are `sgcCharts_Chart` (`TsgcChart`), `sgcCharts_Series` (the line, area, bar and scatter series) and `sgcCharts_VCL` (`TsgcVCLChart`). When you drop the components on a form, Delphi adds these units for you.

Axes

The axes of a chart are the items of its `Axes` collection. A new chart has two: one named `X` at the bottom (`Position = capBottom`) and one named `Y` on the left (`Position = capLeft`). A series has two string properties, `AxisX` and `AxisY`, with the names of the axes it uses. Left empty, as in all five demos, they mean

the first bottom axis and the first left axis.

The properties you will use first:

Property	What it does
AxisType	catNumeric (the default), catLogarithmic, catDateTime OR catCategory .
Title	The text next to the axis, such as "Sales (\$)".
Categories	The names of a category axis, one per line: Jan, Feb, Mar...
LabelFormat	The format of the tick labels. Empty means automatic.
Automatic, Minimum, Maximum	Automatic = True (the default) takes the range from the data. Set it to False to use Minimum and Maximum as given.

On a **category axis** the X of a point is the index of its category: 0 is the first name, 1 the second, and so on. On a **date-time axis** the X of a point is a `TDateTime` . On a **numeric axis** it is just a number.

`LabelFormat` is a `FormatFloat` mask on numeric axes, such as `'#,##0'` , and a date mask on date-time axes, such as `'mmm'` . Both are evaluated *locale invariant*: a numeric axis always uses a dot for decimals and a comma for thousands, and a date axis writes English month names, whatever the Windows regional settings are. On a date mask, `mm` is always the month and `nn` the minutes.

From code, `FindAxis` returns an axis by its name. The enumeration values such as `catCategory` are declared in the unit `sgcCharts_Types` , and `TsgcChartAxis` in `sgcCharts_Axis` , so add both to the uses clause:

```
procedure TForm1.SetupAxes;
var
  oAxis: TsgcChartAxis;
begin
  oAxis := FChart.FindAxis('X');
  oAxis.AxisType := catCategory;
  oAxis.Categories.CommaText := 'North,South,East';

  oAxis := FChart.FindAxis('Y');
  oAxis.Title := 'Orders';
  oAxis.Automatic := False;
  oAxis.Minimum := 0;
  oAxis.Maximum := 25;
end;
```

Adding points

The line, area, bar, scatter and fast line series share the same two methods to add a point:

- `AddXY(X, Y)` appends a point with both coordinates. Use it for numeric and date-time axes.
- `AddY(Y)` appends a point whose X is the X of the newest point plus one, or 0 when the series is empty. Use it for category axes: the first `AddY` lands on the first category, the second on the second, and so on.

`Clear` removes every point. The other series types have methods that suit their data, such as `AddOHLC` for candles. A series with `Color = 0` (the default) takes the next colour of the chart palette, so you only set `Color` when you want a specific one.

When the chart repaints

You never call `Refresh`. Any change to the chart or to one of its series, a new point, a new title, a hidden series, notifies the view, and the view schedules a repaint. Repaints are paced: a view paints at most `Chart.MaxFPS` frames per second (60 by default), so a series that receives a thousand points a second still costs at most 60 paints.

When you make many changes at once, bracket them with `BeginUpdate` and `EndUpdate` on the chart. The notifications are held back and sent once at the end:

```
Chart.BeginUpdate;
try
  for i := 0 to 11 do
  begin
    serLastYear.AddY(CS_LAST_YEAR[i]);
    serThisYear.AddY(CS_THIS_YEAR[i]);
    serTarget.AddY(CS_TARGET[i]);
  end;
finally
  Chart.EndUpdate;
end;
```

This excerpt is from demo 02. Always use `try` and `finally`, so an exception cannot leave the chart in an update that never ends.

Buffers, capacity and the time window

Every series keeps its points in a *ring buffer*, available as its `Buffer` property. Two series properties control it:

- `Capacity`: how many points the series keeps, 100,000 by default.
- `BufferMode`: what happens when the buffer is full. `cbmOverwrite` (the default) drops the oldest point to make room for the new one. `cbmGrow` doubles the capacity instead.

For a real time chart, the chart has a third property, `TimeWindow`. It is a number of seconds, 0 by default (off). When it is set, an automatic numeric or date-time bottom axis shows only the last `TimeWindow` seconds of X, ending at the newest point, so the chart scrolls by itself as new points arrive. `Capacity` limits the memory, `TimeWindow` limits what is on screen; demo 03 uses both.

Large series are also *decimated* before they are drawn: the chart reduces the visible points to about one set per pixel column, keeping the minimum and the maximum of each column, so a line of a million points looks exact and draws fast.

Data sources

Instead of calling `AddXY` yourself, you can let a data source do it. A data source is a non visual component with a `Series` property: point it to a series and every value the source reads becomes a point of that series.

Data source	Reads from
TsgcChartDataSource_DataSet	Any TDataSet: XField and YField name the fields. With Live = True the chart follows every post, insert, delete and filter change. Demo 05 uses it.
TsgcChartDataSource_Manual	Your own code: AppendValue , AppendJSON , AppendCategory . Safe to call from a background thread.
TsgcChartDataSource_JSON , _CSV	A JSON document or CSV text.
TsgcChartDataSource_Table	Any sgcUI table, such as an sgcGrid.

Data sources notify the chart through a *throttle*: the `Throttle` property, in milliseconds, coalesces a burst of new values into one repaint per window.

Threads

The rule is simple. `AddXY` , `AddY` and every property of the chart, the series and the view belong to the main thread. When your data arrives in a background thread, do not call `AddXY` there. Give the series a `TsgcChartDataSource_Manual` and call its `AppendValue` from the thread instead: every append goes through the lock of the buffer, and the data source asks the chart to repaint in the main thread.

```

procedure TFeedThread.Execute;
var
  i: Integer;
begin
  i := 0;
  while not Terminated do
    begin
      { AppendValue is safe from any thread: it goes through the buffer lock }
      FSource.AppendValue(i, Random(100));
      Inc(i);
      Sleep(50);
    end;
end;

procedure TForm1.StartFeed;
var
  oLine: TsgcChartSeries_Line;
begin
  oLine := TsgcChartSeries_Line.Create(Self);
  oLine.Title := 'Sensor';
  oLine.Chart := FChart;

  FSource := TsgcChartDataSource_Manual.Create(Self);
  FSource.Throttle := 100;
  FSource.Series := oLine;

  FFeed := TFeedThread.Create(FSource);
end;

```

`TsgcChartDataSource_Manual` is declared in the unit `sgcCharts_DataSource` . From Delphi 2010 the repaint notification reaches the main thread through `TThread.Queue` . Delphi 7 to 2009 have no `TThread.Queue` , so there the data source notifies in the appending thread; the VCL view handles that safely by posting a message

to its own window, but your own handler of the data source events must synchronize before it touches a control. A timer, as in demo 03, needs none of this, because a `TTimer` fires in the main thread.

Mouse, events and the hit record

The view handles the mouse for you. The settings are in `Chart.Interaction` :

Property	Default	What it does
<code>Crosshair</code>	<code>True</code>	Lines that follow the mouse over the plot.
<code>Tooltip</code>	<code>True</code>	The value of the point under the mouse.
<code>ZoomMode</code>	<code>czmX</code>	Which axes zoom: <code>czmNone</code> , <code>czmX</code> , <code>czmY</code> OR <code>czmXY</code> .
<code>WheelZoom</code> , <code>BoxZoom</code> , <code>Pan</code>	<code>True</code>	Zoom with the wheel, zoom into a box dragged with the left button, pan with the right or middle button.
<code>SnapToPoint</code>	<code>True</code>	Snap the crosshair to the nearest point.

A double click resets the zoom, and so does a call to `Chart.Interaction.ResetZoom` . Ctrl with a left drag pans, and Shift with the wheel pans horizontally.

The chart raises three events you will use early: `OnSeriesClick` when the user clicks a point, `OnCrosshairMove` while the mouse moves over the plot, and `OnZoom` after a zoom or a pan. The first two are of type `TsgcChartHitEvent` and receive a `TsgcChartHitInfo` record (unit `sgcCharts_Interaction`) with the point the mouse is on: `Series` , `SeriesIndex` , `Index` (the point), `XValue` and `YValue` . `Series` is typed `TComponent` and is `nil` when no point is near, so check it and cast it to `TsgcChartSeries` before you use it. Demo 04 shows all three events.

Themes and colours

`Chart.Theme` picks the background, text, grid and axis colours and the series palette in one step. Its type, `TsgcUITheme` , comes from `sgcUICore` (unit `sgcUICore_Theme`):

- `utDefault` , the default, follows the theme of the application set through `sgcUICore`, so all `sgcUI` controls change together.
- `utLight` and `utDark` fix the chart to a light or a dark look.
- `utHighContrast` uses the high contrast colours.
- `utCustom` keeps `BackgroundColor` , `PlotBackgroundColor` and `Palette` exactly as you set them. Setting one of those properties switches the theme to `utCustom` automatically.

The `Palette` holds eight colours. A series with `Color = 0` takes the palette colour of its `Index` , so in the light theme the first series is blue, the second orange, the third green, and so on.

Summary: who does what

You want to...	Use
Set the title, the legend, the axes, the theme	<code>TsgcChart : Title , Subtitle , Legend , Axes , Theme</code>
Show the chart on a form, export or print it	<code>TsgcVCLChart : SaveToPNG , SaveToSVG , Print , ExportToPDF</code>
Draw values as a line, bars, a pie...	A series component with its <code>chart</code> property set
Add points from code	<code>AddXY</code> and <code>AddY</code> in the main thread
Add points from a thread	<code>TsgcChartDataSource_Manual.AppendValue</code>
Show a dataset	<code>TsgcChartDataSource_DataSet</code> with <code>Live = True</code>
Scroll a live chart	<code>Chart.TimeWindow</code> plus <code>Series.Capacity</code>
React to the mouse	<code>OnSeriesClick</code> , <code>OnCrosshairMove</code> , <code>OnZoom</code> and <code>Chart.Interaction</code>

4. The QuickStart demos

This chapter walks through the five QuickStart demos in order. Each section has the same parts: what you will build, the components on the form, the code, how it works, and a few changes to try. Open the demo in Delphi before you read its section.

The code is copied from the demo sources. Where a listing leaves something out, a `// ...` comment marks the place. The header comment of every `.pas` file explains the demo again, so the sources are worth reading on their own.

4.1 Demo 01: Hello Chart

Folder `01.HelloChart`, project `HelloChart.dpr`.

What you will build

The smallest sgcCharts application: one line with the number of website visitors of fourteen days, a title, and two axes that set their range by themselves.



Figure 4.1. Demo 01. The legend is hidden because there is only one series; the series title still appears in the tooltip.

Components on the form

Component	Class	Properties you set
VCLChart	TsgcVCLChart	Chart = Chart, Align = alClient
Chart	TsgcChart	Title.Text = 'Website visitors per day', Legend.Visible = False
serVisits	TsgcChartSeries_Line	Chart = Chart, Title = 'Visitors'

That is the whole design. The chart keeps its two default axes, and the series uses them because `AxisX` and `AxisY` are empty.

The code

The form unit uses three `sgcCharts` units, one for each kind of component:

```
uses
  Classes, Controls, Forms,
  // sgc
  sgcCharts_Chart, sgcCharts_Series, sgcCharts_VCL;
```

The data is a constant array, and `FormCreate` adds one point per day:

```
const
  { visitors of each day, day 1 first }
  CS_VISITS: array [0 .. 13] of Integer = (420, 515, 480, 610, 590, 380, 350,
    640, 720, 690, 810, 780, 520, 490);

procedure TfrmHelloChart.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  { one AddXY per point: X is the day (1 to 14), Y the visitors. The chart
    repaints by itself, there is no Refresh to call. }
  for i := Low(CS_VISITS) to High(CS_VISITS) do
    serVisits.AddXY(i + 1, CS_VISITS[i]);
end;
```

How it works

1. When the form loads, `VCLChart.Chart` links the view to the chart, and `serVisits.Chart` adds the line to the chart.
2. `FormCreate` calls `AddXY` fourteen times. Each call appends a point to the buffer of the series and notifies the chart.
3. The view receives the notifications and schedules one repaint. When it paints, each automatic axis takes its range from the data: the X axis covers days 1 to 14 and the Y axis the visitor counts, both with round tick values.
4. The mouse support is already on: crosshair, tooltip, wheel zoom, box zoom and pan. A double click resets the zoom.

Try this

- Change the class of `serVisits` to `TsgcChartSeries_Bar`, in the form class and in the form file (View as Text). The same points become bars.
- Set `serVisits.Smooth := True` for a curved line, or `serVisits.Markers.Visible := True` to draw a marker on every point.
- Set `Chart.Theme := utDark`. In code, add `sgcUICore_Theme` to the uses clause first.

4.2 Demo 02: Chart Setup

Folder `02.ChartSetup` , project `ChartSetup.dpr` .

What you will build

A monthly sales chart configured almost entirely in the Object Inspector: month names along a category axis, two grouped bar series for last year and this year, a dashed target line with markers, a subtitle, a formatted value axis and the legend at the bottom.

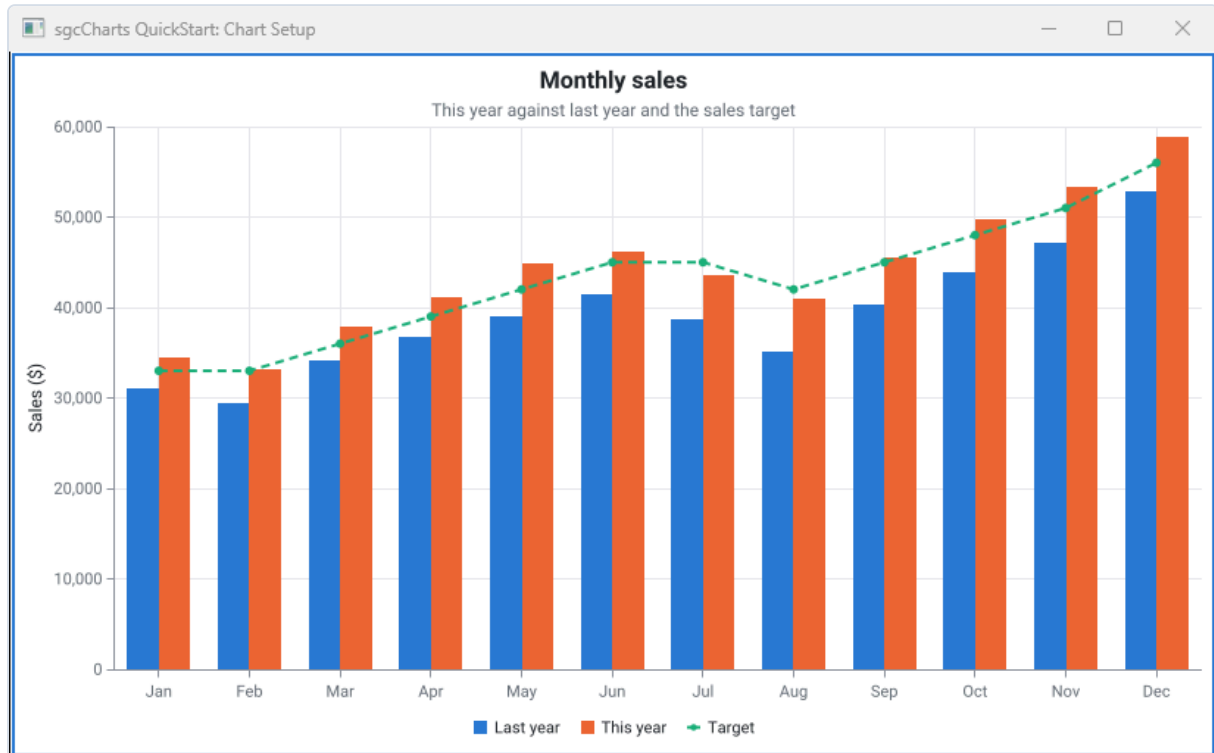


Figure 4.2. Demo 02. The two bar series sit side by side in every month because both use `Grouping = cbgGrouped` .

Components on the form

Component	Class	Properties you set
VCLChart	TsgcVCLChart	Chart = Chart , Align = alClient
Chart	TsgcChart	Title.Text = 'Monthly sales' , Subtitle.Text , Legend.Position = clpBottom , two items in Axes (below)
serLastYear	TsgcChartSeries_Bar	Title = 'Last year' , Grouping = cbgGrouped
serThisYear	TsgcChartSeries_Bar	Title = 'This year' , Grouping = cbgGrouped
serTarget	TsgcChartSeries_Line	Title = 'Target' , LineWidth = 2 , Dash = cdsDash , Markers.Visible = True

Every series also has `Chart = Chart` . The two axes are edited in the collection editor of `Axes` ; in the form file they read:

```
Axes = <
  item
    Name = 'X'
    AxisType = catCategory
```

```

        Position = capBottom
        Categories.Strings = (
            'Jan'
            'Feb'
            // ...
            'Dec')
    end
item
    Name = 'Y'
    Title = 'Sales ($)'
    LabelFormat = '#,##0'
end>

```

The code

The monthly values are three constant arrays, `CS_LAST_YEAR`, `CS_THIS_YEAR` and `CS_TARGET`, of twelve `Double` values each.

```

procedure TfrmChartSetup.FormCreate(Sender: TObject);
var
    i: Integer;
begin
    { BeginUpdate and EndUpdate bracket many changes, so the chart is painted
      once at the end and not after every point. }
    Chart.BeginUpdate;
    try
        for i := 0 to 11 do
            begin
                serLastYear.AddY(CS_LAST_YEAR[i]);
                serThisYear.AddY(CS_THIS_YEAR[i]);
                serTarget.AddY(CS_TARGET[i]);
            end;
        finally
            Chart.EndUpdate;
        end;
    end;
end;

```

How it works

1. The X axis is a category axis with twelve names. On a category axis, the X of a point is the index of its category.
2. `AddY` appends a point at X = 0 for the first call, then 1, 2 and so on, so the twelve values of each series land on Jan to Dec without any X in the code.
3. The Y axis has no `Position` in the form file because `capLeft` is the default. Its `LabelFormat = '#,##0'` writes the ticks with a thousands separator, always a comma, whatever the regional settings.
4. The two bar series use `Grouping = cbgGrouped` and the same axes, so they are placed side by side. The line series is drawn over them.
5. `BeginUpdate` and `EndUpdate` hold back 36 notifications and send one, so the chart is painted once. The legend lists the three series in the order they joined the chart.

Try this

- Set `Grouping = cbgStacked` on both bar series: this year sits on top of last year.
- Set `Orientation = cboHorizontal` on both bar series for horizontal bars.
- Set `Legend.Position = clpRight`, or `Legend.Visible = False`.

4.3 Demo 03: Chart Live Feed

Folder `03.ChartLiveFeed`, project `ChartLiveFeed.dpr`.

What you will build

A simulated server monitor: a CPU line and a memory area with a gradient, five readings a second, and only the last 30 seconds on screen. The chart scrolls by itself, and a check box pauses the feed.

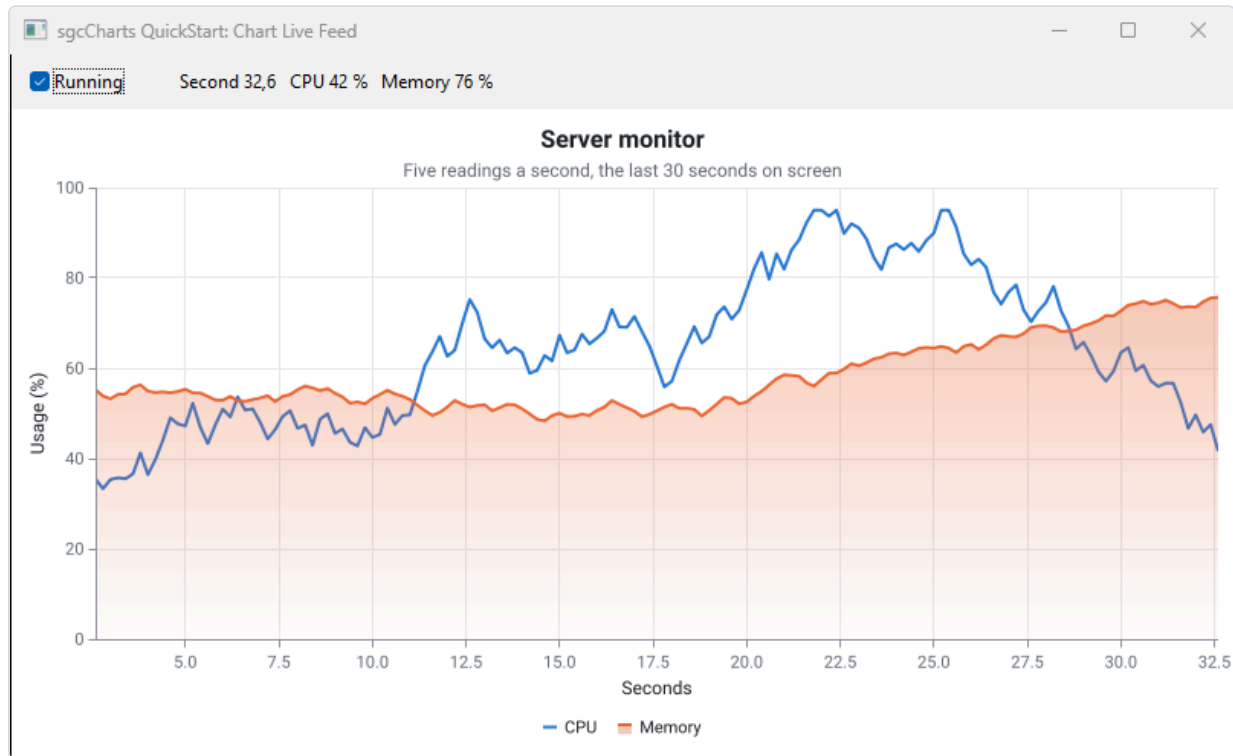


Figure 4.3. Demo 03. The Y axis is fixed from 0 to 100, and the X axis always shows the last 30 seconds.

Components on the form

Component	Class	Properties you set
<code>pnlTop</code>	<code>TPanel</code>	<code>Align = alTop</code> . Holds the check box and the label.
<code>chkRunning</code>	<code>TCheckBox</code>	<code>Caption = 'Running'</code> , <code>Checked = True</code> , <code>OnClick = chkRunningClick</code>
<code>lblLast</code>	<code>TLabel</code>	Shows the last reading.
<code>VCLChart</code>	<code>TsgcVCLChart</code>	<code>Chart = Chart</code> , <code>Align = alClient</code>
<code>Chart</code>	<code>TsgcChart</code>	<code>TimeWindow = 30</code> , <code>Legend.Position = clpBottom</code> , axes below
<code>serCPU</code>	<code>TsgcChartSeries_Line</code>	<code>Title = 'CPU'</code> , <code>Capacity = 1000</code> , <code>LineWidth = 2</code>
<code>serMemory</code>	<code>TsgcChartSeries_Area</code>	<code>Title = 'Memory'</code> , <code>Capacity = 1000</code> , <code>Gradient = True</code>
<code>tmrFeed</code>	<code>TTimer</code>	<code>Interval = 200</code> , <code>OnTimer = tmrFeedTimer</code>

```

Axes = <
  item
    Name = 'X'
    Position = capBottom
    Title = 'Seconds'
  end
  item
    Name = 'Y'
    Title = 'Usage (%)'
    Automatic = False
    Minimum = 0.0000000000000000
    Maximum = 100.00000000000000
  end>
// ...
TimeWindow = 30.000000000000000

```

The code

One method makes a reading and adds it to both series. `FormCreate` calls it 150 times to start with 30 seconds of history, and the timer calls it five times a second:

```

const
  { seconds between two readings, the Interval of tmrFeed }
  CS_STEP = 0.2;

// ...

procedure TfrmChartLiveFeed.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  { a fixed seed, so every run starts with the same picture }
  RandSeed := 2026;
  FCPU := 35;
  FMemory := 55;
  { 30 seconds of history at five readings a second }
  Chart.BeginUpdate;
  try
    for i := 1 to 150 do
      DoAddReading;
    finally
      Chart.EndUpdate;
    end;
  end;
end;

procedure TfrmChartLiveFeed.DoAddReading;
var
  vX: Double;
begin
  { a simulated reading: the last value plus a small random step }
  FCPU := DoClamp(FCPU + (Random - 0.5) * 12, 5, 95);
  FMemory := DoClamp(FMemory + (Random - 0.5) * 3, 40, 80);
  vX := FTick * CS_STEP;
  Inc(FTick);
  { one point per series. With TimeWindow set, the bottom axis follows the
    newest X, and Capacity drops the oldest points when the buffer is
    full. }
  serCPU.AddXY(vX, FCPU);
  serMemory.AddXY(vX, FMemory);
  lblLast.Caption := Format('Second %.1f   CPU %.0f %%   Memory %.0f %%',

```

```

    [vX, FCPU, FMemory]);
end;

procedure TfrmChartLiveFeed.tmrFeedTimer(Sender: TObject);
begin
    DoAddReading;
end;

procedure TfrmChartLiveFeed.chkRunningClick(Sender: TObject);
begin
    tmrFeed.Enabled := chkRunning.Checked;
end;

```

`DoClamp` is a small helper in the unit that keeps a value between a minimum and a maximum.

How it works

1. `X` is the time in seconds: the reading number times 0.2. Both series get a point with the same `X` on every reading.
2. `TimeWindow = 30` makes the bottom axis show the range from the newest `X` minus 30 to the newest `X`. As new points arrive, the window moves and the chart scrolls.
3. `Capacity = 1000` keeps the last 1,000 points of each series, 200 seconds of data. Older points are dropped, because the default `BufferMode` is `cbmOverwrite`, so the memory stays constant however long the demo runs.
4. The Y axis has `Automatic = False` with a range of 0 to 100. Without it, the axis would follow the data and jump up and down as the values change.
5. The timer fires in the main thread, so calling `AddXY` from `tmrFeedTimer` is safe. For a feed thread, see the threads section in chapter 3.

Note. Zoom or pan while the feed runs and the chart stops scrolling, so you can look at the moment you zoomed into. A double click resets the zoom and the scrolling resumes. Without a double click, the scrolling resumes by itself after `Chart.Interaction.AutoScrollResumeDelay` milliseconds without mouse interaction (3000 by default).

Try this

- Set `TimeWindow = 10` for a faster scroll, or 0 to see every point in the buffer.
- Set `Capacity = 50` on both series: only the last ten seconds are kept, and the line no longer reaches the left edge.
- Change `serCPU` to `TsgcChartSeries_FastLine`, the series made for very large and very fast data.

4.4 Demo 04: Chart Events

Folder `04.ChartEvents` , project `ChartEvents.dpr` .

What you will build

A bar chart of units sold by product, online and in store, that reacts to the mouse: a label at the top names the bar you click, a status line at the bottom names the bar under the mouse, and a Reset zoom button becomes active as soon as you zoom.

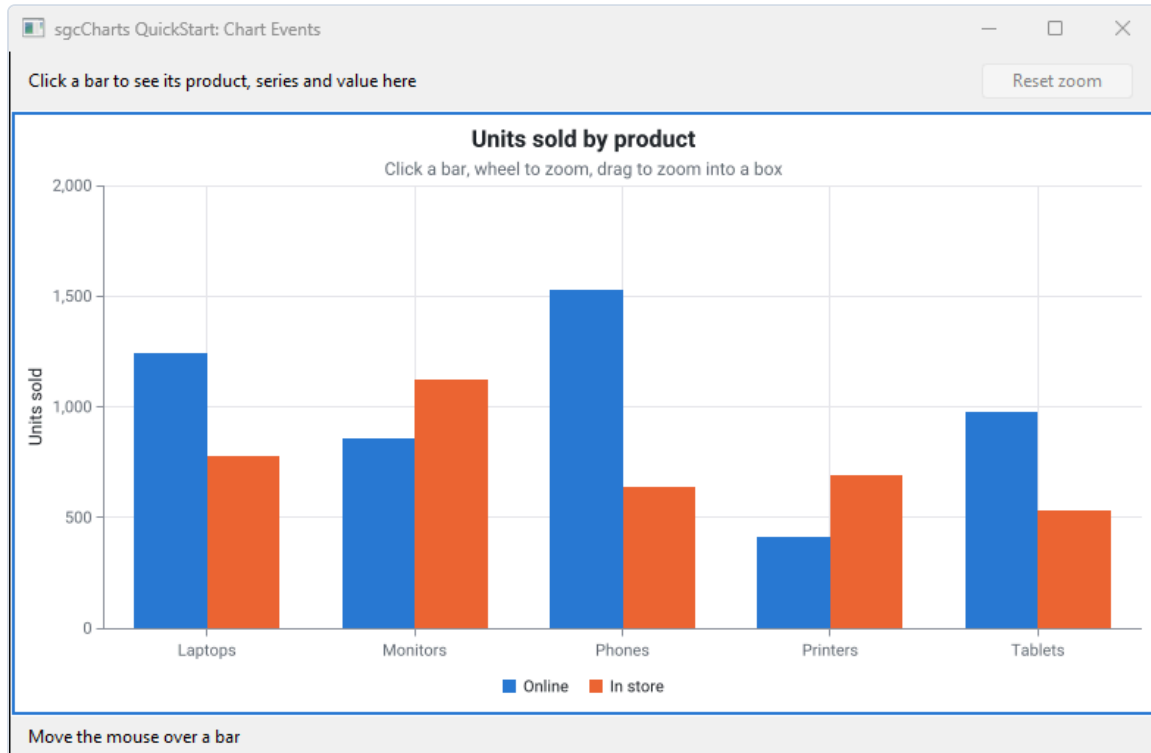


Figure 4.4. Demo 04. Click a bar and the top label shows its product, series and value; the bottom line follows the mouse.

Components on the form

Component	Class	Properties you set
<code>pnlTop</code>	<code>TPanel</code>	<code>Align = alTop</code> . Holds <code>lblClicked</code> and <code>btnReset</code> .
<code>btnReset</code>	<code>TButton</code>	<code>Caption = 'Reset zoom'</code> , <code>Enabled = False</code> , <code>OnClick = btnResetClick</code>
<code>pnlBottom</code>	<code>TPanel</code>	<code>Align = alBottom</code> . Holds <code>lblHover</code> .
<code>VCLChart</code>	<code>TsgcVCLChart</code>	<code>Chart = Chart</code> , <code>Align = alClient</code>
<code>Chart</code>	<code>TsgcChart</code>	X axis <code>catCategory</code> with Laptops, Monitors, Phones, Printers, Tablets; Y axis <code>Title = 'Units sold'</code> , <code>LabelFormat = '#,##0'</code> ; <code>Interaction.ZoomMode = czmXY</code> ; <code>OnSeriesClick</code> , <code>OnCrosshairMove</code> , <code>OnZoom</code>
<code>serOnline</code> , <code>serStore</code>	<code>TsgcChartSeries_Bar</code>	<code>Title = 'Online'</code> and <code>'In store'</code> , <code>Grouping = cbgGrouped</code>

The events are set on the `Chart` component, not on the view: select `Chart` in the designer and double click the events in the Object Inspector.

The code

The handlers need two more units than the previous demos: `sgcCharts_Axis` for `TsgcChartAxis` and `sgcCharts_Interaction` for `TsgcChartHitInfo`:

```
uses
  Classes, SysUtils, Controls, Forms, StdCtrls, ExtCtrls,
  // sgc
  sgcCharts_Axis, sgcCharts_Chart, sgcCharts_Series, sgcCharts_Interaction,
  sgcCharts_VCL;
```

One helper turns a hit record into text such as "Laptops, Online: 1,240 units":

```
function TfrmChartEvents.DoPointText(const aHit: TsgcChartHitInfo): string;
var
  vCategory: string;
  vIndex: Integer;
  oAxis: TsgcChartAxis;
begin
  Result := '';
  { aHit.Series is a TComponent, nil when there is no point near }
  if not (aHit.Series is TsgcChartSeries) then
    Exit;
  { on a category axis the X of a point is the index of its category }
  vCategory := '';
  oAxis := TsgcChartSeries(aHit.Series).XAxis;
  vIndex := Round(aHit.XValue);
  if (oAxis <> nil) and (vIndex >= 0) and (vIndex < oAxis.Categories.Count)
  then
    vCategory := oAxis.Categories[vIndex];
  Result := Format('%s, %s: %s units', [vCategory,
    TsgcChartSeries(aHit.Series).Title, FormatFloat('#,##0', aHit.YValue)]);
end;
```

The four event handlers are short:

```

procedure TfrmChartEvents.ChartSeriesClick(Sender: TObject;
  const aHit: TsgcChartHitInfo);
begin
  if aHit.Series <> nil then
    lblClicked.Caption := 'Clicked: ' + DoPointText(aHit);
end;

procedure TfrmChartEvents.ChartCrosshairMove(Sender: TObject;
  const aHit: TsgcChartHitInfo);
begin
  if aHit.Series <> nil then
    lblHover.Caption := 'Under the mouse: ' + DoPointText(aHit)
  else
    lblHover.Caption := 'Move the mouse over a bar';
end;

procedure TfrmChartEvents.ChartZoom(Sender: TObject);
begin
  btnReset.Enabled := True;
end;

procedure TfrmChartEvents.btnResetClick(Sender: TObject);
begin
  Chart.Interaction.ResetZoom;
  btnReset.Enabled := False;
end;

```

`FormCreate` fills the two series with `AddY` inside `BeginUpdate` and `EndUpdate`, exactly like demo 02.

How it works

1. `OnSeriesClick` and `OnCrosshairMove` are both `TsgcChartHitEvent` events. The `aHit` record describes the point under the mouse: its series, its index, `XValue` and `YValue`.
2. `aHit.Series` is typed `TComponent`, and it is `nil` when the mouse is not near a point. The helper checks it with `is` and casts it to `TsgcChartSeries` to read its `Title` and its `XAxis`.
3. On a category axis `XValue` is the index of the category, so `Categories[Round(aHit.XValue)]` is the product name. The range check keeps the code safe if the index is outside the list.
4. `Interaction.ZoomMode = czmXY` lets the wheel and the box zoom act on both axes; the default `czmX` zooms only horizontally. After every zoom or pan the chart raises `OnZoom`, which enables the button.
5. The button calls `Chart.Interaction.ResetZoom`, the same reset as a double click on the chart.

Note. `OnZoom` also fires after a reset. When you reset with a double click instead of the button, the Reset zoom button is enabled again although there is nothing to reset. It is harmless here; in your own code, compare the axis ranges if you need to know whether the chart is zoomed.

Try this

- Set `Interaction.Tooltip = False` or `Interaction.Crosshair = False` and see what remains.
- Set `Interaction.ZoomMode = czmNone` to turn zooming off.
- In `ChartSeriesClick`, hide the clicked series with `TsgcChartSeries(aHit.Series).Visible := False`. The value axis adjusts to the series that remains.

4.5 Demo 05: Chart Sales App

Folder `05.ChartSalesApp` , project `ChartSalesApp.dpr` .

What you will build

A small sales application: twelve months of revenue and costs in a `TClientDataSet` , edited in a `TDBGrid` with a `TDBNavigator` , and a chart that follows every posted change without a line of chart code. A combo box switches the theme between light, dark and high contrast, and a button exports the chart to a PNG file.

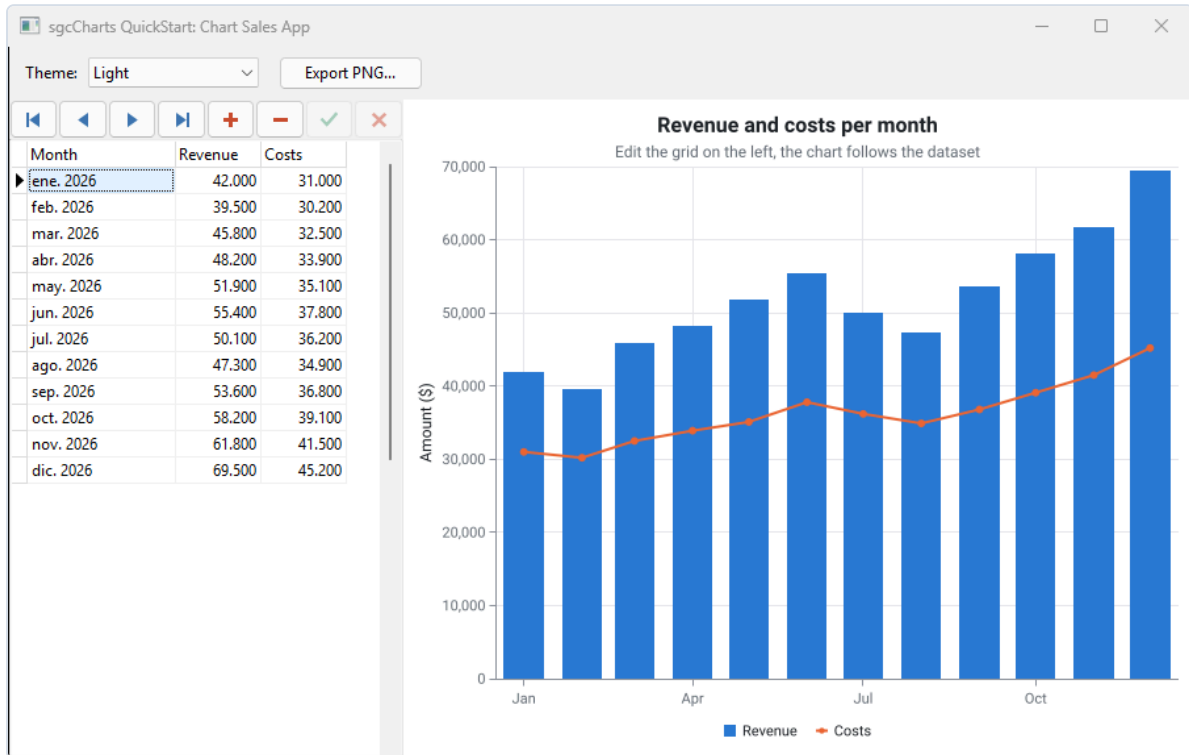


Figure 4.5. Demo 05. The grid shows the dataset in the regional format of the machine; the chart axis uses its own locale invariant format.

Components on the form

Component	Class	Properties you set
cdsSales	TClientDataSet	FieldDefs: Month (ftDateTime), Revenue and Costs (ftFloat)
dsSales	TDataSource	DataSet = cdsSales
navSales , grdSales	TDBNavigator , TDBGrid	DataSource = dsSales , in pnlLeft
cboTheme	TComboBox	Items Light, Dark, High contrast, OnChange = cboThemeChange
btnExport , dlgSave	TButton , TSaveDialog	OnClick = btnExportClick ; DefaultExt = 'png'
VCLChart	TsgcVCLChart	Chart = Chart , Align = alClient
Chart	TsgcChart	Theme = utLight ; X axis catDateTime with LabelFormat = 'mmm' ; Y axis Title = 'Amount (\$)' , LabelFormat = '#,##0'
serRevenue	TsgcChartSeries_Bar	Title = 'Revenue'
serCosts	TsgcChartSeries_Line	Title = 'Costs' , LineWidth = 2 , Markers.Visible = True
srcRevenue , srcCosts	TsgcChartDataSource_DataSet	Series , DataSet = cdsSales , XField = 'Month' , YField , Live = True

The two data sources are the heart of the demo. In the form file, without their positions:

```

object srcRevenue: TsgcChartDataSource_DataSet
    Series = serRevenue
    DataSet = cdsSales
    XField = 'Month'
    YField = 'Revenue'
    Live = True
end
object srcCosts: TsgcChartDataSource_DataSet
    Series = serCosts
    DataSet = cdsSales
    XField = 'Month'
    YField = 'Costs'
    Live = True
end

```

The code

The uses clause adds the dataset units, `MidasLib` (so no `midas.dll` is needed), the theme unit of `sgcUICore` and the data source units:

```

uses
    Classes, SysUtils, Controls, Forms, Dialogs, StdCtrls, ExtCtrls, Grids,
    DBGrids, DBCtrls, DB, DBClient, MidasLib,
    // sgc
    sgcUICore_Theme, sgcCharts_Chart, sgcCharts_Series, sgcCharts_DataSource,
    sgcCharts_DataSource_DataSet, sgcCharts_VCL;

```

`FormCreate` only fills the dataset. There is no chart code in it:

```

procedure TfrmChartSalesApp.FormCreate(Sender: TObject);
var
  vYear, vMonth, vDay: Word;
  i: Integer;
begin
  { the fields are defined in the DFM (FieldDefs); CreateDataSet opens the
    empty in-memory table, and the two Live data sources load at once }
  cdsSales.CreateDataSet;
  TDateTimeField(cdsSales.FieldByName('Month')).DisplayFormat := 'mmm yyyy';
  TFloatField(cdsSales.FieldByName('Revenue')).DisplayFormat := '#,##0';
  TFloatField(cdsSales.FieldByName('Costs')).DisplayFormat := '#,##0';

  { twelve months of the current year. DisableControls holds back the
    notifications, EnableControls sends one, so the chart reloads once. }
  DecodeDate(Date, vYear, vMonth, vDay);
  cdsSales.DisableControls;
  try
    for i := 1 to 12 do
      cdsSales.AppendRecord([EncodeDate(vYear, i, 1), CS_REVENUE[i],
        CS_COSTS[i]]);
    cdsSales.First;
  finally
    cdsSales.EnableControls;
  end;
end;

```

The theme and the export are one line each:

```

procedure TfrmChartSalesApp.cboThemeChange(Sender: TObject);
begin
  case cboTheme.ItemIndex of
    1:
      Chart.Theme := utDark;
    2:
      Chart.Theme := utHighContrast;
  else
      Chart.Theme := utLight;
  end;
end;

procedure TfrmChartSalesApp.btnExportClick(Sender: TObject);
begin
  if dlgSave.Execute then
    VCLChart.SaveToPNG(dlgSave.FileName);
end;

```

How it works

1. Each data source points to one series (`Series`) and one dataset (`DataSet`). For every record, the value of `XField` becomes the X of a point and the value of `YField` its Y. Because `Month` is a date-time field, X is a `TDateTime`, which is why the X axis is a `catDateTime` axis.
2. `Live = True` links the source to the dataset. Opening the dataset loads the chart, closing it clears the chart, and every post, insert, delete or filter change reloads it.
3. Reloads are throttled: the first change reloads at once, and further changes within the `Throttle` window are combined into one reload.
4. Edits that are not posted yet are ignored. Type a new revenue in the grid and the chart changes when you leave the row or press the Post button of the navigator, not while you type.

5. `Chart.Theme` repaints the chart with the colours of the chosen theme. The grid and the other VCL controls keep their look, because they are standard controls.
6. `SaveToPNG` renders the chart into a PNG file at the size of the control.

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

Try this

- Set `srcRevenue.FollowCursor = True`. Moving through the grid now focuses the matching bar in the chart.
- Replace `SaveToPNG` with `SaveToSVG` (and the dialog filter), or call `VCLChart.ShowPrintPreview`.
- Set `Chart.Theme = utDark` at design time and see the designer repaint.

5. Main components

The SGC Charts palette page is long, but a first application needs only four or five of its components: the chart, the view, one or two series and perhaps a data source. This chapter introduces them, then points to the components you will want later.

Every class has its own help topic, named after the class, such as `TsgcChart.htm` or `TsgcChartSeries_Bar.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
<code>TsgcChart</code>	SGC Charts	The chart model	<code>Title</code> , <code>Subtitle</code> , <code>Legend</code> , <code>Axes</code> , <code>Theme</code> , <code>Palette</code> , <code>TimeWindow</code> , <code>Interaction</code> , <code>BeginUpdate</code> , <code>EndUpdate</code> , <code>OnSeriesClick</code>
<code>TsgcVCLChart</code>	SGC Charts	The VCL control that shows a chart	<code>Chart</code> , <code>SaveToPNG</code> , <code>SaveToSVG</code> , <code>Print</code> , <code>ShowPrintPreview</code> , <code>ExportToPDF</code> , <code>ShowEditor</code>
<code>TsgcFMXChart</code>	SGC Charts (FireMonkey)	The FireMonkey control	<code>Chart</code>
<code>TsgcChartRenderer</code>	SGC Charts	Renders a chart without a window, to PNG or SVG	See its help topic
<code>TsgcChartSeries_Line</code>	SGC Charts	A line through the points	<code>AddXY</code> , <code>AddY</code> , <code>LineWidth</code> , <code>Dash</code> , <code>Markers</code> , <code>Smooth</code>
<code>TsgcChartSeries_FastLine</code>	SGC Charts	A plain line for very large or very fast data	<code>AddXY</code> , <code>LineWidth</code>
<code>TsgcChartSeries_Area</code>	SGC Charts	A line with the area below it filled	<code>AddXY</code> , <code>Gradient</code>
<code>TsgcChartSeries_Bar</code>	SGC Charts	Vertical or horizontal bars	<code>AddY</code> , <code>Grouping</code> , <code>Orientation</code>
<code>TsgcChartSeries_Pie</code>	SGC Charts	A pie or a donut	See its help topic
<code>TsgcChartDataSource_DataSet</code>	SGC Charts	Fills a series from a <code>TDataSet</code>	<code>Series</code> , <code>DataSet</code> , <code>XField</code> , <code>YField</code> , <code>Live</code> , <code>FollowCursor</code> , <code>Throttle</code>
<code>TsgcChartDataSource_Manual</code>	SGC Charts	Fills a series from your code, from any thread	<code>Series</code> , <code>AppendValue</code> , <code>AppendJSON</code> , <code>AppendCategory</code> , <code>Throttle</code>
<code>TsgcChartAxis</code>	none, an item of <code>Chart.Axes</code>	One axis	<code>Name</code> , <code>AxisType</code> , <code>Position</code> , <code>Title</code> , <code>Categories</code> , <code>LabelFormat</code> , <code>Automatic</code> , <code>Minimum</code> , <code>Maximum</code>

Class	Palette page	What it is for	Key members
<code>TsgcChartInteraction</code>	none, it is <code>Chart.Interaction</code>	Mouse settings	<code>Crosshair</code> , <code>Tooltip</code> , <code>ZoomMode</code> , <code>WheelZoom</code> , <code>BoxZoom</code> , <code>Pan</code> , <code>ResetZoom</code>

The chart and its views

TsgcChart

The component you drop first, and the one you configure most. It holds everything that is not data: the `Title` and `Subtitle`, the `Legend` (with `Visible` and `Position`), the `Axes` collection, the `Theme` and the eight colour `Palette`, the `Margins`, the `TimeWindow` of a live chart and the `Interaction` settings. It also raises the mouse events. It is a non visual component, so it can live on a data module and be shown by a view on another form.

TsgcVCLChart

The control on the form. Set its `Chart` property, align it, and it does the rest: it paints, it handles the mouse and it repaints when the chart changes, at most `Chart.MaxFPS` times a second. It is also where the output methods are: `SaveToPNG` and `SaveToSVG` write the chart at the size of the control, `Print` and `ShowPrintPreview` print it, `ExportToPDF` writes a PDF, and `ShowEditor` opens a chart editor for the user at run time.

TsgcFMXChart and TsgcChartRenderer

`TsgcFMXChart` is the FireMonkey twin of the VCL control, for Windows, macOS, iOS and Android.

`TsgcChartRenderer` needs no form at all: it renders a `TsgcChart` to PNG or SVG in a console program, a service or a web server, on Windows and on Linux64. The `Headless` and `HeadlessService` demos use it.

The series

All series join a chart through `Chart` and share `Title`, `Color`, `Visible`, `ShowInLegend`, `AxisX` and `AxisY`. They differ in the data they take and how they draw it:

- **Line family:** `TsgcChartSeries_Line`, `_FastLine`, `_Area` and `_StepLine`. X and Y points, added with `AddXY` or `AddY`.
- **Bars and points:** `TsgcChartSeries_Bar` (grouped or stacked, vertical or horizontal), `_Scatter` and `_Bubble`.
- **Financial:** `TsgcChartSeries_Candlestick` and `_OHLC` take candles with open, high, low and close. The financial pack adds indicator series computed from a candle series, a volume series and an aggregator that builds candles from trades.
- **Categories:** `TsgcChartSeries_Pie`, `_Radar`, `_Funnel`, `_Treemap` and `_Heatmap` work with named items rather than an X axis.
- **Business:** `TsgcChartSeries_BoxPlot`, `_Waterfall` and `_RangeArea`.

The data sources

A data source fills one series. Set its `Series` property, then tell it where the data is:

- `TsgcChartDataSource_DataSet` : a `TDataSet` , with `XField` and `YField` , plus `OpenField` to `VolumeField` for candles and `LabelField` for category series. `Live = True` follows the dataset, `FollowCursor = True` focuses the point of the current record.
- `TsgcChartDataSource_Manual` : values you append in code, from any thread, with `AppendValue` , `AppendOHLC` , `AppendJSON` or `AppendCategory` .
- `TsgcChartDataSource_JSON` and `TsgcChartDataSource_CSV` : a JSON document or CSV text, with a `Mapping` that names the X and Y fields or columns.
- `TsgcChartDataSource_Table` and `TsgcChartTableLink` : any sgcUI table, for example an `sgcGrid`, as the data of a chart.

Beyond the basics

Once the QuickStart demos feel familiar, these are worth a look. Each one has a help topic and most have a demo:

- **More axes**: several value axes on one chart, each series bound to one by name through `AxisY` (demo `MultiAxis`).
- **Annotations and error bars**: lines, bands and labels on the plot, and error bars on a series (demo `BusinessCharts`).
- **The runtime editor**: `VCLChart.ShowEditor` lets the user change the chart while the application runs.
- **Printing and PDF**: `Print` , `ShowPrintPreview` and `ExportToPDF` on the view.
- **Live feeds**: the `sgcChartsWS` bridge with data sources for WebSocket, MQTT, Server-Sent Events and Binance (demos `WebSocketFeed` , `MQTTSensors` , `SSEFeed` , `BinanceLive`).
- **The financial pack**: indicators, panes, drawing tools and the trade aggregator (demos `Candlestick` , `DrawingTools`).

6. Common problems

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

Compiling

Undeclared identifier for an enumeration value or a type

What you see. Code that sets a property compiles fine in the Object Inspector but fails in code with `E2003 Undeclared identifier`, for example on `catCategory`, `utDark` or `TsgcChartAxis`.

Why. The form unit gets the units of the components you dropped, but the enumeration types and the helper classes live in other units, which Delphi does not add for you.

Fix. Add the unit that declares the identifier:

Identifier	Unit
<code>catNumeric</code> , <code>catDateTime</code> , <code>catCategory</code> , <code>capLeft</code> , <code>capBottom</code>	<code>sgcCharts_Types</code>
<code>TsgcChartAxis</code>	<code>sgcCharts_Axis</code>
<code>clpTop</code> , <code>clpBottom</code> , <code>clpRight</code> (legend)	<code>sgcCharts_Chart</code>
<code>cbgGrouped</code> , <code>cbgStacked</code> , <code>cboHorizontal</code> , <code>cdsDash</code>	<code>sgcCharts_Series</code>
<code>TsgcChartHitInfo</code> , <code>czmXY</code>	<code>sgcCharts_Interaction</code>
<code>cbmOverwrite</code> , <code>cbmGrow</code>	<code>sgcCharts_Buffer</code>
<code>utLight</code> , <code>utDark</code> , <code>utHighContrast</code>	<code>sgcUICore_Theme</code>

aHit.Series has no Title

What you see. In `OnSeriesClick` or `OnCrosshairMove`, `aHit.Series.Title` does not compile.

Why. `TsgcChartHitInfo.Series` is typed `TComponent`, and it is `nil` when the mouse is not near a point.

Fix. Check it and cast it: `if aHit.Series is TsgcChartSeries then ... TsgcChartSeries(aHit.Series).Title`, as the helper of demo 04 does.

ExportToPDF raises "PDF export needs sgcPDF"

What you see. `VCLChart.ExportToPDF` raises `EsgcUIPDFExportUnavailable` with the message "PDF export needs sgcPDF: add the unit `sgcUIPDF_Export` to the project".

Why. The PDF writer is not part of `sgcCharts`. It comes from `sgcPDF`, and it is only linked when your project uses it.

Fix. Add `sgcUIPDF_Export` to the uses clause of a unit of the project. It needs `sgcPDF` to be installed. `SaveToPNG`, `SaveToSVG` and `Print` need nothing extra.

An exception at start up after the trial period

What you see. An application that worked during the evaluation now raises an exception when it starts.

Why. It was built with the trial version, and the trial period is over.

Fix. Install the Registered version and build the application again.

Data and axes

The bars are one category to the right

What you see. On a category axis, the first value is drawn on the second category, and the last one sits after the last category.

Why. On a category axis the X of a point is the index of its category, and indexes start at 0. `AddXY(1, ...)` puts the point on the second name.

Fix. Use `AddY`, which starts at 0 and adds one per call, or pass 0-based X values to `AddXY`.

The axis labels ignore my regional settings

What you see. On a Spanish or German Windows, the value axis still shows `60,000` and the date axis shows English month names, while a `TDBGrid` next to it uses your locale.

Why. This is by design. `LabelFormat` is evaluated locale invariant, so a chart looks the same on every machine and in every exported file.

Fix. None is needed. Keep in mind that in a date mask `mm` is always the month and `nn` the minutes.

The value axis jumps up and down

What you see. In a live chart the Y axis changes its range every few seconds, which makes the lines look as if they move.

Why. An axis with `Automatic = True` takes its range from the data that is visible, and that data changes.

Fix. When you know the range, set `Automatic = False` with `Minimum` and `Maximum`, as demo 03 does for a percentage.

The last bar touches the edge of the plot

What you see. With monthly bars on a `catDateTime` axis, as in demo 05, the last bar sits right at the right edge of the plot.

Why. The automatic range of a date-time axis ends at the last X, and a bar is centred on its X.

Fix. It is only cosmetic. If it bothers you, set `Automatic = False` on the X axis with a `Maximum` a little after the last date, or use a category axis with the month names.

The dataset chart stays empty

What you see. A `TsgcChartDataSource_DataSet` is on the form, but the series shows no points.

Why. The source loads only from an open dataset, and only when `Live = True` (or when you call `LoadFromDataSet`). A row whose X or Y field is null gives no point, and an `XField` or `YField` that names no field gives nothing. `Live` does nothing at design time; the link is made when the form loads.

Fix. Check `Series`, `DataSet`, `XField`, `YField` and `Live`, and make sure the dataset is open at run time. An empty `XField` is allowed: X is then the record index.

The chart does not change while I type in the grid

What you see. You type a new value in the `TDBGrid` and the chart keeps the old one.

Why. Edits that are not posted are ignored, and nothing reloads while the dataset is in edit state.

Fix. Post the record: leave the row, press Enter, or press the Post button of the navigator.

Live data and interaction

The live chart stopped scrolling

What you see. A chart with `TimeWindow` no longer follows the newest points.

Why. You zoomed or panned. A user zoom freezes the automatic scroll, so the user can look at what they zoomed into.

Fix. Double click the chart, or call `Chart.Interaction.ResetZoom`. The scroll also resumes by itself after `Interaction.AutoScrollResumeDelay` milliseconds without interaction (3000 by default; 0 means never).

The oldest points disappear

What you see. A series that receives many points loses its first ones.

Why. The buffer of the series is full. `Capacity` is 100,000 by default, and with `BufferMode = cbmOverwrite` each new point replaces the oldest.

Fix. Raise `Capacity`, or set `BufferMode = cbmGrow` to let the buffer double when it is full.

Adding points from a background thread

What you see. Your data arrives in a background thread, and calling `AddXY` from that thread gives unreliable results.

Why. `AddXY` and `AddY` are made for the main thread. The data sources are the components made for feed threads: their appends go through the buffer lock, and their repaint notification reaches the main thread.

Fix. Bind a `TsgcChartDataSource_Manual` to the series and call its `AppendValue` from the thread, as shown in chapter 3. On Delphi 7 to 2009, synchronize any of your own event handlers of the data source before they touch a control.

Zoom only works horizontally

What you see. The wheel and the box zoom change the X range but never the Y range.

Why. `Interaction.ZoomMode` is `czmX` by default.

Fix. Set `ZoomMode = czmXY` (or `czmY`), as demo 04 does.

7. Where next

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

The other Delphi demos

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

Demo	What it shows
LiveLine	A FastLine series with a one million point ring buffer, fed from a background thread at up to 200,000 points per second. The natural next step after demo 03.
DataSetChart	A <code>TClientDataSet</code> with 1,000 rows feeds a line series and a bar series through <code>TsgcChartDataSource_DataSet</code> , next to a <code>DBGrid</code> . The next step after demo 05.
MultiAxis	Three line series on three value axes, with a date-time bottom axis, the legend on the right, and zoom and pan.
Themes	The same chart in the light and the dark theme side by side, a render scale selector, and export to PNG and SVG.
PieDonut	A pie and a donut fed from a table you edit in a string grid, with Others grouping, explode on click, four label modes and PNG and SVG export.
RadarCompare	Three radar series compare three products on eight criteria, filled or as outlines, while a timer moves the scores.
TreemapFunnel	A three level treemap of sales that drills on click, next to a sales funnel, both updated live.
HeatmapLive	A rolling spectrogram on a date-time axis and a region by month matrix with value labels and a colour scale.
BusinessCharts	Bubble, box plot, waterfall and range area series, error bars, annotations, the ISA-101 theme, the runtime chart editor and a live dataset.
Candlestick	One year of daily OHLCV data from an embedded CSV in a candlestick series, with the volume as bars on a second axis.
DrawingTools	Candles in three panes with Bollinger bands, an SMA, the volume and the RSI, and tools to draw trend lines, levels and Fibonacci retracements.
Headless	A console program that renders two charts to PNG and SVG files with <code>TsgcChartRenderer</code> , for Windows and Linux64.
HeadlessService	A Windows service that adds a value to a line chart and renders it to a PNG file every 10 seconds.
WebSocketFeed, MQTTSensors, SSEFeed, BinanceLive	Live charts fed by the sgcChartsWS data sources from a WebSocket server, an MQTT broker, a Server-Sent Events stream and the Binance market data.
BusinessChartsFMX, FMXMobile	FireMonkey: the business series demo, and a live FastLine plus a candlestick chart with pinch zoom and pan on touch screens.

Some of these demos use features or products that are not part of every installation, such as the financial pack or sgcWebSockets. Their readme files say what they need.

Help topics

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

Topic	What it covers
Introduction	What sgcCharts does and where to start.
Architecture	The buffers, the decimation, the chart, series and axes, the data sources and the bridge.
Platforms	The Delphi versions, the target platforms and the packages.
TsgcChart, Axes, Title, legend and margins	The chart model in depth.
TsgcChartSeries, TsgcChartBuffer	What every series shares, and the ring buffer.
TsgcChartDataSource_DataSet, TsgcChartDataSource_Manual	The two data sources of this guide.
TsgcChartInteraction	Crosshair, tooltip, zoom, pan and the hit record.
Themes and palette, Printing	The look of the chart and its output.
One topic per class	<code>TsgcVCLChart</code> , every series and every data source, with each property, its type and its default.

On the website

- <https://www.esegece.com/products/sgccharts/>: 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 sgcCharts version and edition, and the demo or a small project that shows the problem.

Enjoy turning your data into pictures.

sgcCharts and sgcUI are products of eSeGeCe. <https://www.esegece.com>