sgcGauges

Getting Started

Dials, meters, LEDs and alarms 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 live values on it with sgcGauges: a speedometer, a thermometer, a tank level, a row of LEDs or a knob that the user turns. You do not need any experience with gauge components. If you have ever placed a `TProgressBar` on a form and set its `Position`, you already know the basic idea.

The guide follows the five QuickStart demos. Each demo adds one idea to the previous one, from a single dial moved by a track bar to a small pump station monitor with input gauges, an alarm and a PNG snapshot. 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 controls and watch the needles, then read its section and try the suggested changes. Chapters 1 to 3 explain what sgcGauges 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\Gauges\Delphi\00.QuickStart` folder of your sgcUI installation. Each demo has its own subfolder, from `01.HelloGauge` to `05.GaugeMonitorApp`, with a Delphi project (`.dpr`) that you can open and run directly. All of them are VCL applications for Windows.

The screenshots were taken on a machine with Spanish regional settings. The gauges format their own numbers with a point as the decimal separator on every machine (chapter 3 explains how to change that), while text that a demo formats itself, such as the times in the alarm log, follows the Windows regional settings of the machine running the demo.

Contents

1. What sgcGauges is

The idea in one minute
Your first gauge needs one line of code
What you need
Part of sgcUI
Coming from TGauge and TProgressBar

2. Install and the palette

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

3. Core concepts

A control and a model
Scales, pointers and Value
Ranges and colours
Feeding values and animation
Thresholds and OnThreshold
Input gauges
Themes and skins
Text and number format
Threads
Summary: who does what

4. The QuickStart demos

4.1 Demo 01: Hello Gauge
4.2 Demo 02: Gauge Scales
4.3 Demo 03: Gauge Alarms
4.4 Demo 04: Gauge Skins
4.5 Demo 05: Gauge Monitor App

5. Main components

Reference table
Display gauges
Input gauges
Data and live feeds
The model classes

6. Common problems

Values and pointers
Alarms and events
Look and text
Editions and compiling

7. Where next

The other Delphi demos
Help topics
On the website
Getting help

1. What sgcGauges is

sgcGauges adds gauges to Delphi applications, for the VCL and for FireMonkey. A gauge shows a number the way an instrument does: a needle on a dial, a filled column on a thermometer, a ring that closes as a download progresses, a row of LED segments, a seven segment display, a lamp that lights up when something is wrong. Four input gauges, a knob, a slider, a toggle switch and a push button, let the user set values in the same visual style.

You drop a gauge such as `TsgcVCLRadialGauge` on a form, set its scale, title and units in the Object Inspector, and write numbers to its `Value` property while the application runs. The gauge draws itself, animates the needle to each new value, colours the zones of the scale and raises an event when a value crosses a limit you defined.

The idea in one minute

Every gauge, whatever it looks like, is built from the same three parts:

- **A scale**: the range of values the gauge can show, from `Minimum` to `Maximum`, with its ticks, its labels and optional coloured zones called *ranges*, for example green up to 6 bar, amber up to 8 and red above.
- **A pointer**: the thing that shows a value on the scale. On a dial it is a needle, on a linear gauge it can be a filled bar or a marker, on a progress ring it is the arc.
- **The text around it**: a `Title` such as “Speed”, the `Units` such as “km/h” and the value as text, formatted with `ValueFormat`.

A new gauge already has one scale from 0 to 100 and one pointer, so it works the moment you drop it on a form. `Value` is the value of that first pointer. Write a number to it and the needle moves there. That is the entire idea; everything else, such as several scales, thresholds, skins or input gauges, is detail.

Your first gauge needs one line of code

The first QuickStart demo is the smallest gauge application there is: a radial gauge and a track bar. The gauge is configured at design time, and the form file stores only what differs from the defaults:

```
object Gauge: TsgcVCLRadialGauge
  Left = 0
  Top = 0
  Width = 440
  Height = 372
  Pointers = <
    item
      Value = 65.000000000000000000
    end>
  Title = 'Speed'
  Units = 'km/h'
  Align = alClient
end
```

And this is the only event handler of the demo, the `OnChange` of the track bar:

```
procedure TfrmHelloGauge.tbSpeedChange(Sender: TObject);
begin
  // Value is a Double: the gauge accepts any number inside its scale and
  // clamps the ones outside it
  Gauge.Value := tbSpeed.Position;
end;
```

Run it, drag the track bar, and the needle follows with a short animation. Chapter 4 explains the demo in detail.

What you need

- **Delphi**: any version from Delphi 7 to Delphi 13. There are packages for Delphi 7, 2007, 2009, 2010, XE to XE8, 10 Seattle to 10.4 Sydney, 11, 12 and 13.
- **VCL**: the `TsgcVCL...Gauge` controls run on Windows, Win32 and Win64, in every supported Delphi version. The QuickStart demos use the VCL.
- **FireMonkey**: the `TsgcFMX...Gauge` controls, with the same properties, from Delphi XE6 on, on the FireMonkey platforms.
- **No window at all**: the gauge models also run without a form, in a console program or a service, and write PNG and SVG images. From Delphi 10.2 on this works on Linux64 too.
- **sgcUICore**: the shared foundation of the sgcUI products. It is installed together with sgcGauges.

Part of sgcUI

sgcGauges is one product of sgcUI, the eSeGeCe suite of visual components. All sgcUI products share sgcUICore, which gives them the same themes (`utDefault` , `utLight` , `utDark` , `utCustom` and `utHighContrast`), the same colour type and the same drawing engine. A gauge with `Theme = utDark` matches a dark sgcGrid or sgcDashboard on the same form.

Two bridges connect sgcGauges to other products. With sgcDashboard, gauges can live inside dashboard tiles and follow the dashboard theme. With sgcWebSockets, the sgcGaugesWS bridge feeds gauges from WebSocket, MQTT and server sent events without code. You do not need either of them for this guide.

Coming from TGauge and TProgressBar

Many Delphi applications show a value with the controls that come with Delphi: a `TProgressBar` , a `TTrackBar` , or the old `TGauge` from the Samples page. They are fine for a percentage. sgcGauges is meant for the cases where a value is a measurement: it has a real scale with any minimum and maximum, labelled ticks, coloured zones, units, thresholds that raise events with hysteresis, smooth animation, themes and skins, and it scales cleanly with the size of the control. The way you use it stays the same as with a progress bar: set a property, and the control repaints.

2. Install and the palette

This chapter gets sgcGauges 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

sgcGauges 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 **sgcGauges**. The installer shows one checkbox per product and enables the products your licence includes. sgcUICore has no checkbox: it is installed automatically with sgcGauges.
4. Optionally select **sgcGauges for sgcWebSockets**, the bridge with the live feed sources. It installs only when sgcWebSockets is present, and you do not need it for this guide.
5. The installer copies the files, compiles the packages on your machine for each selected Delphi version and installs the design time packages.

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

The packages

You only need this section if you build the packages yourself. sgcGauges 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
`sgcGauges<V>`	Runtime	The types, the models, the painter, the skins and `TsgcGaugeDataLink`, with the PNG and SVG output. No VCL, no FireMonkey.
`sgcGaugesVCL<V>`	Runtime	The VCL controls, unit `sgcGauges_VCL`.
`sgcGaugesFMX<V>`	Runtime, XE6 and later	The FireMonkey controls, unit `sgcGauges_FMX`.
`dclsgcGauges<V>`	Design time	Registers the VCL gauges and `TsgcGaugeDataLink` on the palette.
`dclsgcGaugesFMX<V>`	Design time, XE6 and later	Registers the FireMonkey gauges.
`sgcGaugesWS<V>`, `dclsgcGaugesWS<V>`	Runtime and design time	The sgcWebSockets bridge with the WebSocket, MQTT and SSE feed sources.

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

The palette page

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

Component	What it shows
`TsgcVCLRadialGauge`	A dial with a needle: a full dial, a half dial, a quarter dial or a custom arc.
`TsgcVCLLinearGauge`	A straight scale, horizontal or vertical: a bar, a thermometer or a bullet graph.
`TsgcVCLDigitalGauge`	A seven segment display, such as an odometer or a clock face.
`TsgcVCLLEDGauge`	A single lamp that is on or off, optionally blinking.
`TsgcVCLLEDBarGauge`	A row of LED segments, coloured by the ranges of the scale.
`TsgcVCLProgressRingGauge`	A ring that fills as the value grows, with an optional percentage.
`TsgcVCLTankGauge`	A tank with a liquid level.
`TsgcVCLSymbolGauge`	A process symbol, such as a pump or a valve, for HMI panels.
`TsgcVCLKnobGauge`	Input: a rotary knob the user turns.
`TsgcVCLSliderGauge`	Input: a slider.
`TsgcVCLToggleSwitchGauge`	Input: an on and off switch.
`TsgcVCLPushButtonGauge`	Input: a push button, momentary or latching, with an LED.
`TsgcGaugeDataLink`	Non visual: binds a gauge to a dataset field.
`TsgcVCLVUMeterGauge`, `TsgcVCLClockGauge`, `TsgcVCLCompassGauge`, `TsgcVCLAnnunciatorGauge`	A VU meter, an analogue clock, a compass and an alarm annunciator panel. Not in the Community edition.

In a FireMonkey project the same page shows the `TsgcFMX...` versions of these controls. With the sgcWebSockets bridge installed, the feed sources `TsgcGaugeFeedSource_WebSocket`, `TsgcGaugeFeedSource_MQTT` and `TsgcGaugeFeedSource_SSE` are added as well. The QuickStart demos use the radial, linear, digital, LED, LED bar and progress ring gauges, the knob and the toggle switch.

Editing gauges at design time

A gauge draws itself in the form designer, so what you see while you edit is what the user sees at run time. The parts of a gauge are collections, edited with the usual Delphi collection editor:

- Select the gauge, click the ellipsis button of `Scales` in the Object Inspector, and select the first item to set `Minimum`, `Maximum`, `MajorInterval`, `LabelFormat` and the rest. Its `Ranges` property opens a second collection editor for the coloured zones.
- `Pointers` works the same way: the first item holds the starting value, the colour and the `Kind` of the pointer.

- `Thresholds` holds the alarm limits.

> **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.

Editions

sgcGauges is built from the same source for every edition. The Registered edition includes everything, and the Trial edition, meant for evaluation, has the same features. The Community edition leaves out the advanced features:

- A gauge draws, animates and checks thresholds only for its **first scale and first pointer**. Further scales and pointers stay in their collections, so a form made with the Registered edition still loads.
- The **VU meter, clock, compass and annunciator** are not available.
- The **14 segment and dot matrix** displays of the digital gauge draw as 7 segment.
- The **sgcGaugesWS bridge** is not included, and `TsgcGaugeDataLink` reads a field but does not write user changes back to it.

Everything else is in every edition: the radial, linear, digital, LED, LED bar, progress ring, tank and symbol gauges, the four input gauges, ranges, thresholds, animation, themes, skins, printing and the PNG and SVG export.

> **Note.** The QuickStart demos use one scale and one pointer per gauge and none of the advanced gauge types, so they compile and behave the same in every edition.

Where the demos are

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

Folder	Project	What it shows
01.HelloGauge	`HelloGauge.dpr`	One radial gauge moved by a track bar
02.GaugeScales	`GaugeScales.dpr`	Scales, coloured ranges, gauge styles, pointer kinds
03.GaugeAlarms	`GaugeAlarms.dpr`	Live values from a timer, a threshold with hysteresis, an LED and a log
04.GaugeSkins	`GaugeSkins.dpr`	Light and dark themes and the built-in skins, switched at run time
05.GaugeMonitorApp	`GaugeMonitorApp.dpr`	A pump station monitor with input gauges, an alarm, a time display and a PNG snapshot

The other folders in `Demos\Gauges\Delphi` are larger demos, listed in chapter 7.

Running your first demo

1. Open `Demos\Gauges\Delphi\00.QuickStart\01.HelloGauge\HelloGauge.dpr` in Delphi.
2. Press **F9**. The project compiles and a window with a speed dial appears, the needle at 65.
3. Drag the track bar at the bottom. The needle follows it and the value text below the needle changes.

3. Core concepts

This chapter explains the ideas you need before the demos make full sense. It is short on purpose: each section is one idea, with the property names you will meet in the Object Inspector. The code samples here are not taken from the demos; each one was compiled in a small test program.

A control and a model

Every gauge has two halves. The **model**, a class such as `TsgcRadialGauge` in a framework neutral unit, holds the configuration and the values and knows how to draw itself. The **control**, such as `TsgcVCLRadialGauge` in the unit `sgcGauges_VCL`, puts a model on a form: it creates the model, publishes its properties in the Object Inspector, repaints when the model changes and runs the animation.

In everyday work you only talk to the control. Its public property `Gauge` returns the model, for the few methods that live there, such as `SetValue`, `ValueToText` or `BeginUpdate`. Because the model needs no window, the same gauges can be rendered to PNG or SVG in a console program or a service.

A gauge is created in code like any other control. The unit to add to the `uses` clause is `sgcGauges_VCL`:

```
procedure AddSpeedGauge(aForm: TForm);
var
  oGauge: TsgcVCLRadialGauge;
begin
  oGauge := TsgcVCLRadialGauge.Create(aForm);
  oGauge.Parent := aForm;
  oGauge.Align := alClient;
  oGauge.Title := 'Speed';
  oGauge.Units := 'km/h';
  // the range first, then the value: a value is clamped to its scale
  oGauge.Scales[0].Maximum := 240;
  oGauge.Scales[0].MajorInterval := 40;
  oGauge.Value := 128;
end;
```

Scales, pointers and Value

A gauge has three collections: `Scales`, `Pointers` and `Thresholds`. A new gauge always has one scale, from 0 to 100, and one pointer, so `Scales[0]` and `Pointers[0]` exist from the start.

- **A scale** (`TsgcGaugeScale`) has `Minimum`, `Maximum`, `MajorInterval` (the distance between labelled ticks), `MinorCount` (the small ticks between two labels, 4 by default), `LabelFormat` for the tick labels, `Reversed` and its `Ranges`.
- **A pointer** (`TsgcGaugePointer`) has a `Value`, a `Color` and a `Kind`: `gpkNeedle` (the default), `gpkMarker`, `gpkBar` or `gpkRange`.

The `Value` property of a gauge is a shortcut for `Pointers[0].Value`. Two details follow from that, and both surprise people at first:

- `Value` itself is not saved in the form file. When you set `Value` in the Object Inspector, the DFM stores it as `Pointers[0].Value`, as you saw in chapter 1.

- A value is clamped to the range of its scale when it is written. With the default scale, 150 becomes 100. Set `Minimum` and `Maximum` first and the value afterwards, as the sample above does.

To set a pointer without animation, or a pointer other than the first, use `SetValue` on the model:

```
procedure ResetGauge(aGauge: TsgcVCLRadialGauge);
begin
  // pointer 0 to 0, no animation
  aGauge.Gauge.SetValue(0, 0, False);
end;
```

> **Note.** In the Community edition only the first scale and the first pointer are drawn, so the QuickStart demos use exactly one of each.

Ranges and colours

A range (`TsgcGaugeRange`) colours a part of a scale, from `StartValue` to `EndValue` : the green, amber and red zones of a dial, or the colours of the segments of an LED bar. Ranges belong to a scale, so you find them under `Scales[0].Ranges` .

Colours in sgcGauges are `TsgcUIColor` values, written as `$AARRGGBB` : alpha, red, green, blue. `$FFE34948` is an opaque red. A colour of `0` means "take the colour from the theme", which is why most colour properties are 0 by default. In the DFM a colour appears as a signed integer, for example `-1881784` for `$FFE34948` ; the Object Inspector shows the same number.

```
procedure AddRedZone(aGauge: TsgcVCLRadialGauge);
var
  oRange: TsgcGaugeRange;
begin
  oRange := aGauge.Scales[0].Ranges.Add;
  oRange.StartValue := 200;
  oRange.EndValue := 240;
  oRange.Color := $FFE34948; // $AARRGGBB: opaque red
end;
```

`TsgcGaugeRange` and `TsgcGaugeThreshold` are declared in the unit `sgcGauges_Model` , so add it to the `uses` clause when your code names them.

Feeding values and animation

A gauge does not fetch its data. Your application writes `Value` whenever it has a new reading: from a timer, from a socket, from a database, from another control. The QuickStart demos use a `TTimer` , because it needs no hardware.

Each new value starts an animation from the current position to the new one. `Animation.Duration` is 300 milliseconds by default, `Animation.Easing` chooses the curve, and `Animation.Enabled := False` (or a duration of 0) makes the pointer jump. There is no animation at design time or for the values loaded from the form file, so a form opens with its needles already in place.

Drawing is cheap when only the value changes. A gauge keeps its static parts (face, scale, ticks, labels, ranges, title) as cached pixels and redraws only the pointers and the value text for a new value. A change of a static property redraws everything. When you change several static properties at once, bracket them with `BeginUpdate` and `EndUpdate` so the gauge repaints once:

```
procedure Restyle(aGauge: TsgcVCLRadialGauge);
begin
  aGauge.Gauge.BeginUpdate;
  try
    aGauge.Title := 'Pressure';
    aGauge.Units := 'bar';
    aGauge.Scales[0].Maximum := 16;
  finally
    aGauge.Gauge.EndUpdate;
  end;
end;
```

Thresholds and OnThreshold

A threshold (`TsgcGaugeThreshold`) watches the value of one pointer and fires the `OnThreshold` event of the gauge when the value crosses it. It is the way to build alarms. Its main properties are `Value` (the limit), `Direction` , `Hysteresis` , `Color` (a mark on the scale when not 0) and `ValueColor` (the colour of the value text while the alarm is active).

- **Direction.** `gtdRising` , the default, is active at or above the limit; `gtdFalling` is active at or below it. With these two, `OnThreshold` fires *only when the alarm starts*. Use `gtdBoth` to get an event when it starts and another when it clears; the `aRising` parameter tells which one.
- **Hysteresis.** A rising alarm at 90 with a hysteresis of 3 clears only when the value drops below 87. Without it, a value that wobbles around 90 would fire again and again.
- **When.** A threshold is evaluated when a value is written, on the new value, not on the positions drawn during the animation. The `Active` property tells whether the alarm is on right now.

```
procedure AddHighAlarm(aGauge: TsgcVCLRadialGauge);
var
  oThreshold: TsgcGaugeThreshold;
begin
  oThreshold := aGauge.Thresholds.Add;
  oThreshold.Value := 90;
  oThreshold.Hysteresis := 3;
  oThreshold.Direction := gtdBoth; // an event when it starts and when it clears
  oThreshold.Color := $FFE34948;   // a red mark on the scale at 90
end;
```

The `gtdBoth` constant is declared in `sgcGauges_Types` . Demo 03 builds the same threshold in the Object Inspector instead.

Input gauges

The knob, the slider, the toggle switch and the push button look like gauges and follow the same themes and skins, but the user changes them with the mouse, the keyboard, touch or the mouse wheel. The knob and the slider keep their value in `Value` , on `Scales[0]` , snapped to `Step` ; the switch and the button expose `Checked` .

Their `OnChange` event fires **only for changes made by the user**, never when your code writes `Value` or `Checked`. You can therefore set an input gauge from its own handler, or from the data it controls, without starting a loop.

Themes and skins

Two properties decide the look of a gauge:

- **`Theme`** picks the colour set of sgcUICore: `utDefault`, `utLight`, `utDark`, `utCustom` or `utHighContrast`. It is the same theme type that the other sgcUI products use.
- **`Skin`** is the name of a registered skin, a named set of visual settings applied on top of the theme: the face, the bezel, the needle, the text weight. The built-in skins are Classic, Modern, Industrial, ISA101 and Dark. An empty `Skin`, the default, uses the default skin of the application, which is none until you set one.

The order is simple: a colour property set on the gauge itself, such as `FaceColor` or `Pointers[0].Color`, always wins; then the skin; then the theme. To give every gauge of the application the same skin, set the default skin once, for example in the `OnCreate` of the main form, with the unit `sgcGauges_Skin` in the `uses` clause:

```
sgcGaugesSetDefaultSkin('Modern');
```

Under the high contrast theme no skin is applied, so the gauges keep the system colours.

Text and number format

`Title`, `Units` and the value text are drawn by the gauge itself. `ValueFormat` formats the value (for example `'0.0'`), `ShowValue` hides it, and on the radial gauge `ValuePosition` moves it to the bottom, the centre or the top. Each scale has its own `LabelFormat` for the tick labels. Font sizes are relative to the size of the gauge, so the text grows and shrinks with the control.

For text that a format string cannot express, handle `OnFormatValue`: it receives the value and the text made with `ValueFormat`, and you replace the text. Demo 05 uses it to show seconds as hours, minutes and seconds.

The gauges format numbers with their own format settings, with a point as the decimal separator, so a gauge shows the same text on every machine whatever its regional settings. To use the separators of the Windows regional settings instead, assign them to the model (this sample uses the global `FormatSettings` variable of Delphi XE and later):

```
procedure UseLocalNumbers(aGauge: TsgcVCLRadialGauge);
begin
  // the separators of the Windows regional settings instead of '.'
  aGauge.Gauge.FormatSettings := FormatSettings;
end;
```

When your code needs the value as the gauge shows it, for a log or a label, call `Gauge.ValueToText`, as demo 03 does.

Threads

`Value`, `SetValue` and the `Value` of a pointer may be written from any thread. The VCL and FireMonkey controls repaint and call `OnThreshold` in the main thread, whatever thread wrote the value, so a worker thread that reads a sensor can write straight to the gauge:

```
procedure TSensorThread.Execute;
begin
  while not Terminated do
  begin
    // Value may be written from any thread; the control repaints
    // and fires OnThreshold in the main thread
    FGauge.Value := ReadTemperature;
    Sleep(500);
  end;
end;
```

Every other property, such as the scale, the colours or the title, belongs to the main thread.

Summary: who does what

Piece	Its job	Where you set it
The control (`TsgcVCLRadialGauge` and the others)	Puts a gauge on a form, repaints, animates, delivers events in the main thread	The form designer
`Gauge` (the model)	Configuration, values, drawing; `SetValue`, `ValueToText`, `BeginUpdate`	Code
`Scales` and `Ranges`	The range of values, ticks, labels and coloured zones	Collection editor, or code
`Pointers`	What shows a value: needle, marker or bar; `Pointers[0].Value` is `Value`	Collection editor, or code
`Thresholds` and `OnThreshold`	Alarm limits with direction and hysteresis	Collection editor and an event handler
`Animation`	How a pointer moves to a new value	Object Inspector
`Theme` and `Skin`	Colours and visual style	Object Inspector, or code
Your code	Writes `Value` when a new reading arrives	A timer, a thread, an event

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 together they cover what most applications need from a gauge.

Demo	Project	New ideas
01.HelloGauge	`HelloGauge.dpr`	A gauge on a form, `Value`, the animation
02.GaugeScales	`GaugeScales.dpr`	Scales, ranges, radial and linear styles, pointer kinds, the progress ring
03.GaugeAlarms	`GaugeAlarms.dpr`	Live values from a timer, a threshold with hysteresis, `OnThreshold`, the LED
04.GaugeSkins	`GaugeSkins.dpr`	Themes and skins at run time, the digital gauge, the LED bar
05.GaugeMonitorApp	`GaugeMonitorApp.dpr`	Input gauges, a simulation loop, `OnFormatValue`, `SaveToPNG`

> **Note.** The demos need sgcGauges installed, so the components appear on the SGC Gauges palette page and the `sgcUI\Source` folder is on the library path. They use one scale and one pointer per gauge, so they behave the same in every edition. Their values come from code or from the form file; they need no network, no files and no hardware.

4.1 Demo 01: Hello Gauge

Folder `01.HelloGauge`, project `HelloGauge.dpr`.

What you will build

The smallest gauge application there is. A radial gauge fills the window and shows a speed in km/h. A track bar at the bottom moves the needle: drag it, and the needle follows with a short animation while the value text below it counts along.



Figure 4.1. Demo 01. The scale from 0 to 100, the ticks, the labels and the value text are all defaults; only the title, the units and the starting value were set.

Components on the form

Component	Class	Properties you set
`Gauge`	`TsgcVCLRadialGauge`	`Align = alClient`, `Title = 'Speed'`, `Units = 'km/h'`, `Pointers[0].Value = 65`.
`pnlBottom`	`TPanel`	`Align = alBottom`. Holds the label and the track bar.
`lblSpeed`	`TLabel`	`Caption = 'Speed:'`.
`tbSpeed`	`TTrackBar`	`Max = 100`, `Position = 65`, `OnChange = tbSpeedChange`.

The gauge needs only four lines in the form file. The value was typed into `Value` in the Object Inspector, but the DFM keeps it as `Pointers[0].Value`:

```
  object Gauge: TsgcVCLRadialGauge
    Left = 0
    Top = 0
    Width = 440
    Height = 372
    Pointers = <
      item
        Value = 65.000000000000000000
      end>
    Title = 'Speed'
    Units = 'km/h'
    Align = alClient
  end
```

The code

The `uses` clause names one sgcGauges unit, the one with the VCL controls:

```
uses
  SysUtils, Classes, Controls, Forms, StdCtrls, ExtCtrls, ComCtrls,
  // sgc
  sgcGauges_VCL;
```

And this is the only event handler of the demo:

```
procedure TfrmHelloGauge.tbSpeedChange(Sender: TObject);
begin
  // Value is a Double: the gauge accepts any number inside its scale and
  // clamps the ones outside it
  Gauge.Value := tbSpeed.Position;
end;
```

How it works

1. The form loads from its DFM. The gauge creates its model with the default scale from 0 to 100 and one pointer, then reads the title, the units and `Pointers[0].Value`. Values loaded from the form file are not animated, so the needle starts at 65.
2. The user drags the track bar, and `tbSpeedChange` writes the new position to `Gauge.Value`.
3. The gauge starts a 300 millisecond animation from the old value to the new one and redraws only the needle and the value text on every frame. The face, the ticks and the labels come from a cached image.

Try this

- Set `Animation.Duration` to 1000 for a slow, heavy needle, or `Animation.Enabled` to `False` to make it jump.
- Set `ValuePosition` to `gvpCenter` or `gvpTop` to move the value text.
- Replace the radial gauge with a `TsgcVCLLinearGauge` or a `TsgcVCLProgressRingGauge` named `Gauge`. The line of code stays the same.

4.2 Demo 02: Gauge Scales

Folder `02.GaugeScales`, project `GaugeScales.dpr`.

What you will build

Three gauges, configured entirely in the Object Inspector: a half dial for a pressure in bar with green, amber and red zones, a vertical thermometer from −20 to 50 °C, and a progress ring for a battery level. A button switches the pressure scale between 10 and 16 bar to show how the ranges follow the scale.

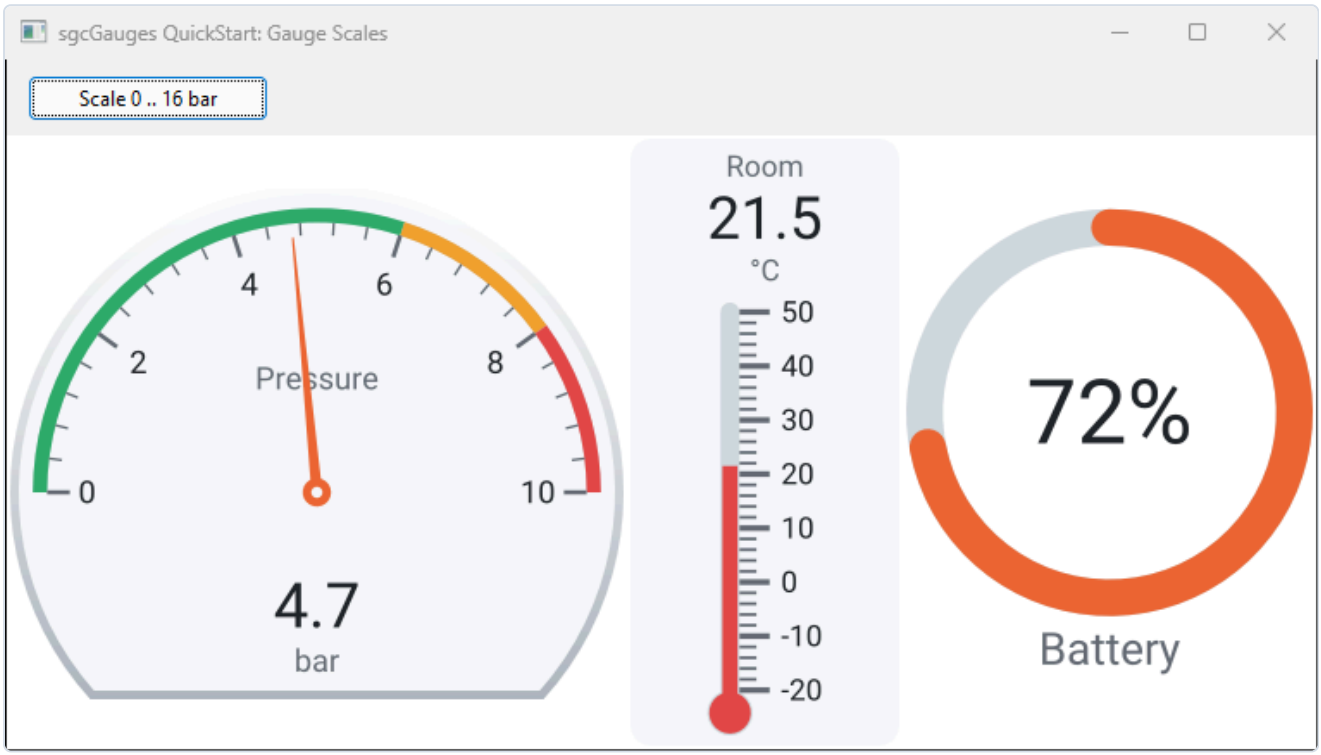


Figure 4.2. Demo 02. Three gauge types, each with one scale and one pointer.

Components on the form

Component	Class	Properties you set
`gPressure`	`TsgcVCLRadialGauge`	`RadialStyle = grsHalf`. `Scales[0]`: `Maximum = 10`, `MajorInterval = 2`, `LabelFormat = '0'`, three `Ranges` (0 to 6 green, 6 to 8 amber, 8 to 10 red). `Pointers[0].Value = 4.7`, `Title = 'Pressure'`, `Units = 'bar'`, `ValueFormat = '0.0'`.
`gTemperature`	`TsgcVCLLinearGauge`	`Orientation = gloVertical`, `LinearStyle = glsThermometer`. `Scales[0]`: `Minimum = -20`, `Maximum = 50`, `MajorInterval = 10`. `Pointers[0]`: `Kind = gpkBar`, a red `Color`, `Value = 21.5`. `Title = 'Room'`, `Units = °C`.
`gBattery`	`TsgcVCLProgressRingGauge`	`Thickness = 0.18`, `RoundCaps = True`, `ShowPercent = True`, `Pointers[0].Value = 72`, `Title = 'Battery'`.
`btnRange`	`TButton`	`OnClick = btnRangeClick`, on the top panel `pnlTop`.

This is the pressure gauge in the form file. The three ranges are items of `Ranges`, inside the first item of `Scales`. The colours are `$AARRGGBB` values written as signed integers; the first range has no `StartValue` line because 0 is the default:

```
object gPressure: TsgcVCLRadialGauge
  Left = 0
  Top = 44
  Width = 360
  Height = 356
  Scales = <
    item
      Maximum = 10.000000000000000000
      MajorInterval = 2.000000000000000000
      LabelFormat = '0'
      Ranges = <
        item
          EndValue = 6.000000000000000000
          Color = -13718165
        end
        item
          StartValue = 6.000000000000000000
          EndValue = 8.000000000000000000
          Color = -1007568
        end
        item
          StartValue = 8.000000000000000000
          EndValue = 10.000000000000000000
          Color = -1881784
        end>
    end>
  Pointers = <
    item
      Value = 4.700000000000000000
    end>
  Title = 'Pressure'
  Units = 'bar'
  ValueFormat = '0.0'
  RadialStyle = grsHalf
  Align = alLeft
end
```

And the thermometer. Note `Kind = gpkBar` on the pointer: it draws the filled red column.

```
  object gTemperature: TsgcVCLLinearGauge
    Left = 360
    Top = 44
    Width = 160
    Height = 356
    Scales = <
      item
        Minimum = -20.000000000000000000
        Maximum = 50.000000000000000000
        MajorInterval = 10.000000000000000000
        LabelFormat = '0'
      end>
    Pointers = <
      item
        Kind = gpkBar
        Value = 21.500000000000000000
        Color = -1881784
      end>
    Title = 'Room'
    Units = #176'C'
    ValueFormat = '0.0'
    Orientation = gloVertical
    LinearStyle = glsThermometer
    Align = alLeft
  end
```

The code

The demo has one event handler. It changes the end of the pressure scale:

```
procedure TfrmGaugeScales.btnRangeClick(Sender: TObject);
begin
  { Scales is a collection: Scales[0] is its first item. Toggle the end of
    the pressure scale between 10 and 16 bar. The red band (8 .. 10) keeps
    its values, so on a 16 bar scale it covers a smaller part of the dial. }
  if gPressure.Scales[0].Maximum = 10 then
  begin
    gPressure.Scales[0].Maximum := 16;
    btnRange.Caption := 'Scale 0 .. 10 bar';
  end
  else
  begin
    gPressure.Scales[0].Maximum := 10;
    btnRange.Caption := 'Scale 0 .. 16 bar';
  end;
end;
```

How it works

1. Every gauge type uses the same building blocks: `Scales`, `Pointers` and, inside a scale, `Ranges`. Only the drawing differs. The style properties, `RadialStyle` for the dial and `Orientation` and `LinearStyle` for the linear gauge, choose the shape.
2. The ranges are values on the scale, not angles or pixels. When the button sets `Maximum` to 16, the gauge redraws its static layer and the zones stay at 0 to 6, 6 to 8 and 8 to 10 bar, which is now a smaller part of the dial.
3. The progress ring has no ticks or labels. `ShowPercent` shows the value as a percentage of its scale.

Try this

- Set `Scales[0].Reversed` to `True` on the pressure gauge, so the scale runs the other way round.
- Set `RadialStyle` to `grsCustom` and play with `StartAngle` and `SweepAngle`. Angles are in degrees, 0 is 3 o'clock and positive angles turn clockwise.
- On the thermometer, set the pointer `Kind` to `gpkMarker` and `MarkerShape` to `gmsDiamond`.

4.3 Demo 03: Gauge Alarms

Folder `03.GaugeAlarms`, project `GaugeAlarms.dpr`.

What you will build

A boiler temperature that changes on its own. A timer writes a new reading every 400 milliseconds, a slow wave between 60 and 100 °C. Above 90 °C an alarm starts: the value text turns red, an LED blinks and a line is added to a log. When the temperature falls below 87 °C the alarm clears, the LED goes off and the log says so.

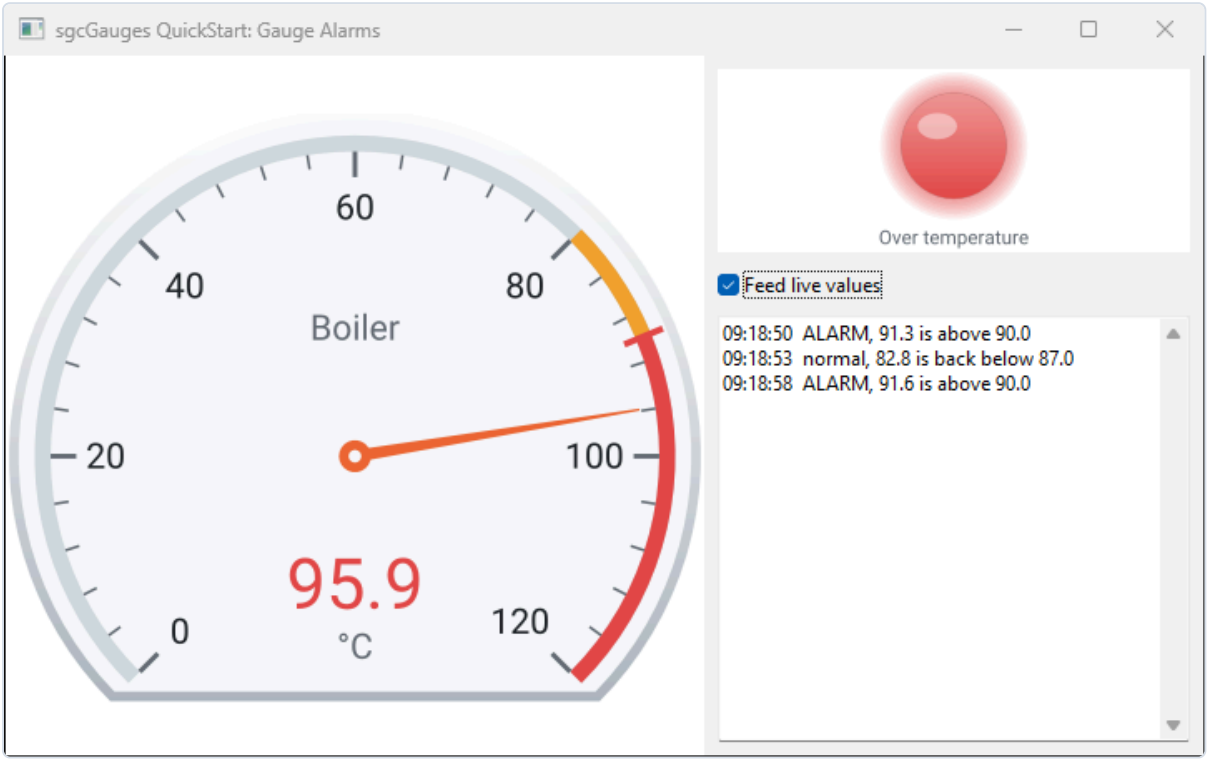


Figure 4.3. Demo 03 during an alarm. The red mark on the scale at 90 is the threshold; the log shows that the alarm cleared at 82.8, below 90 minus the hysteresis of 3.

Components on the form

Component	Class	Properties you set
`gBoiler`	`TsgcVCLRadialGauge`	`Scales[0]`: `Maximum = 120`, `MajorInterval = 20`, ranges 80 to 90 amber and 90 to 120 red. `Thresholds[0]`: `Value = 90`, `Direction = gtdBoth`, `Hysteresis = 3`, red `Color` and `ValueColor`. `Title = 'Boiler'`, `Units = °C`, `ValueFormat = '0.0'`, `OnThreshold = gBoilerThreshold`.
`ledAlarm`	`TsgcVCLLEDGauge`	`Title = 'Over temperature'`, a red `OnColor`, `Blink = True`.
`chkRun`	`TCheckBox`	`Caption = 'Feed live values'`, `OnClick = chkRunClick`.
`memLog`	`TMemo`	The alarm log.
`tmrFeed`	`TTimer`	`Interval = 400`, `OnTimer = tmrFeedTimer`.

The threshold is an item of the `Thresholds` collection of the gauge:

```
    Thresholds = <
      item
        Value = 90.000000000000000000
        Direction = gtdBoth
        Hysteresis = 3.000000000000000000
        Color = -1881784
        ValueColor = -1881784
      end>
    Title = 'Boiler'
    Units = #176'C'
    ValueFormat = '0.0'
    Align = alClient
    OnThreshold = gBoilerThreshold
  end
```

The code

The `uses` clause adds `sgcGauges_Model`, because the `OnThreshold` handler receives a `TsgcGaugeThreshold`:

```
uses
  SysUtils, Classes, Controls, Forms, StdCtrls, ExtCtrls,
  // sgc
  sgcGauges_Model, sgcGauges_VCL;
```

The timer feeds the gauge, and the check box starts and stops the timer:

```
procedure TfrmGaugeAlarms.tmrFeedTimer(Sender: TObject);
begin
  { A new reading every 400 ms: a slow wave between 60 and 100 degrees, so
    the value crosses the limit of 90 every few seconds. }
  Inc(FStep);
  gBoiler.Value := 80 + 20 * Sin(FStep * 0.3);
end;

procedure TfrmGaugeAlarms.chkRunClick(Sender: TObject);
begin
  tmrFeed.Enabled := chkRun.Checked;
end;
```

The threshold handler drives the LED and writes the log:

```
procedure TfrmGaugeAlarms.gBoilerThreshold(Sender: TObject;
  aThreshold: TsgcGaugeThreshold; aValue: Double; aRising: Boolean);
begin
  // the lamp follows the state of the threshold
  ledAlarm.Active := aThreshold.Active;
  { Gauge.ValueToText formats a number exactly like the value text of the
    gauge: ValueFormat '0.0' with a dot as the decimal separator }
  if aRising then
    memLog.Lines.Add(FormatDateTime('hh:nn:ss', Now) + '  ALARM, ' +
      gBoiler.Gauge.ValueToText(0, aValue) + ' is above ' +
      gBoiler.Gauge.ValueToText(0, aThreshold.Value))
  else
    memLog.Lines.Add(FormatDateTime('hh:nn:ss', Now) + '  normal, ' +
      gBoiler.Gauge.ValueToText(0, aValue) + ' is back below ' +
      gBoiler.Gauge.ValueToText(0, aThreshold.Value -
      aThreshold.Hysteresis));
end;
```

How it works

1. Every 400 milliseconds the timer writes a new value. The gauge animates the needle towards it and, at the same moment, checks the threshold against the new value.
2. When the value reaches 90, the threshold becomes active. The gauge colours the value text with `ValueColor` and calls `gBoilerThreshold` with `aRising = True`.
3. The handler copies `aThreshold.Active` to the LED, which starts to blink, and logs the alarm. `ValueToText` formats the numbers exactly like the gauge does.
4. The value goes down again. Between 90 and 87 nothing happens: that is the hysteresis. Below 87 the threshold is no longer active, and because `Direction` is `gtdBoth` the handler is called a second time, with `aRising = False`. The LED goes off and the log says the value is back to normal.

> **Watch out.** With the default `Direction = gtdRising`, `OnThreshold` fires only when the alarm starts, never when it clears, and the LED of this demo would stay on forever. Use `gtdBoth` whenever you need the clear event.

Try this

- Add a second threshold at 70 with `Direction = gtdFalling`, a low temperature alarm. In the handler, `aThreshold.Index` tells the two apart.
- Set `Hysteresis` to 0 and the timer `Interval` to 100, and watch the log fill with alarms around 90.
- Add a check box that switches `gBoiler.Thresholds[0].Enabled` on and off.

4.4 Demo 04: Gauge Skins

Folder `04.GaugeSkins`, project `GaugeSkins.dpr`.

What you will build

A small dashboard with four gauges: a speed dial, a six digit odometer, a horizontal fuel bar and an LED bar for the engine load. Two combo boxes switch the theme (light or dark) and the skin of all four gauges at run time. The form and the top panel follow the theme too.



Figure 4.4. Demo 04 with the Dark theme and the Industrial skin, the starting look. The unlit segments of the odometer and of the LED bar are drawn dimmed.

Components on the form

Component	Class	Properties you set
`gSpeed`	`TsgcVCLRadialGauge`	`Scales[0]`: `Maximum = 240`, `MajorInterval = 40`, a red range from 200 to 240. `Pointers[0].Value = 128`, `Title = 'Speed'`, `Units = 'km/h'`, `ValueFormat = '0'`.
`gOdometer`	`TsgcVCLDigitalGauge`	`Scales[0].Maximum = 999999`, `Pointers[0].Value = 48213`, `Digits = 6`, `ValueFormat = '0'`, `Title = 'Odometer'`, `Units = 'km'`.
`gFuel`	`TsgcVCLLinearGauge`	Horizontal. `Scales[0]`: `Maximum = 60`, `MajorInterval = 10`. `Pointers[0]`: `Kind = gpkBar`, `Value = 41`. `Title = 'Fuel'`, `Units = 'l'`.
`gLevel`	`TsgcVCLLEDBarGauge`	`Segments = 16`, ranges green 0 to 60, amber 60 to 85, red 85 to 100. `Pointers[0].Value = 70`, `Title = 'Engine load'`.
`cboTheme`, `cboSkin`	`TComboBox`	`cboTheme` lists Light and Dark; `cboSkin` is filled in code. Both have `OnChange = cboLookChange`.

Note that the odometer needs `Scales[0].Maximum = 999999`: a digital gauge has a scale like every other gauge, and with the default maximum of 100 the value 48213 would be clamped.

The code

The `uses` clause adds the sgcUICore theme units and the skin registry:

```
uses
  SysUtils, Classes, Controls, Forms, StdCtrls, ExtCtrls,
  // sgc
  sgcUICore_Theme, sgcUICore_Theme_VCL, sgcGauges_Skin, sgcGauges_VCL;
```

`FormCreate` lists the registered skins and selects the starting look:

```
procedure TfrmGaugeSkins.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  // the first item means "no skin"
  cboSkin.Items.Add('(none)');
  for i := 0 to sgcGaugesSkinCount - 1 do
    cboSkin.Items.Add(sgcGaugesSkinName(i));
  cboSkin.ItemIndex := cboSkin.Items.IndexOf('Industrial');
  cboTheme.ItemIndex := 1; // Dark
  DoApplyLook;
end;
```

`DoApplyLook`, called from `FormCreate` and from the `OnChange` of both combo boxes, applies the look to every gauge on the form:

```
procedure TfrmGaugeSkins.DoApplyLook;
var
  i: Integer;
  vTheme: TsgcUITheme;
  vSkin: string;
begin
  if cboTheme.ItemIndex = 1 then
    vTheme := utDark
  else
    vTheme := utLight;
  if cboSkin.ItemIndex <= 0 then
    vSkin := ''
  else
    vSkin := cboSkin.Items[cboSkin.ItemIndex];
  { every VCL gauge descends from TsgcVCLCustomGauge, so one loop over the
    components of the form reaches all of them }
  for i := 0 to ComponentCount - 1 do
    if Components[i] is TsgcVCLCustomGauge then
    begin
      TsgcVCLCustomGauge(Components[i]).Theme := vTheme;
      TsgcVCLCustomGauge(Components[i]).Skin := vSkin;
    end;
  // the form and the top panel follow the theme too
  Color := sgcUIColorToVCL(sgcUIThemeColors(vTheme).Background);
  sgcUIApplyThemeVCL(pnlTop, vTheme);
end;
```

How it works

1. `sgcGaugesSkinCount` and `sgcGaugesSkinName` read the skin registry: first the five built-in skins (Classic, Modern, Industrial, ISA101, Dark), then any skin the application registered itself.
2. All VCL gauges, the input gauges included, descend from `TsgcVCLCustomGauge`, which publishes `Theme` and `Skin`. One loop over `Components` reaches every gauge, whatever its type.
3. The theme gives the base colours. The skin is applied on top of it: face, bezel, needle and text weight. An empty skin name means no skin.
4. The form itself is not a gauge, so the demo colours it with the background colour of the theme, `sgcUIThemeColors(vTheme).Background`, converted to a VCL colour, and themes the top panel with `sgcUIApplyThemeVCL`.

> **Tip.** Not every skin suits every theme. The Industrial and Dark skins are made for a dark form. The Modern skin keeps a light face even under the Dark theme.

Try this

- Remove the skin from the loop and set the `Skin` of just one gauge in the Object Inspector.
- Call `sgcGaugesSetDefaultSkin('Modern')` in `FormCreate`. It applies to every gauge whose `Skin` is empty, so choose `(none)` in the combo box to see it.
- Set the odometer `DisplayType` to `gdt14Segment` or `gdtDotMatrix`. In the Community edition both draw as 7 segment.

4.5 Demo 05: Gauge Monitor App

Folder `05.GaugeMonitorApp`, project `GaugeMonitorApp.dpr`.

What you will build

A small pump station monitor that brings the previous demos together. Water flows into a tank, and a pump takes it out. On the left the user runs the pump with a toggle switch and sets its speed with a knob. In the middle a dial shows the pump flow and a seven segment display the running time as hours, minutes and seconds. On the right a vertical bar shows the tank level, with a high level alarm at 90 %. A button saves the flow dial as a PNG image.

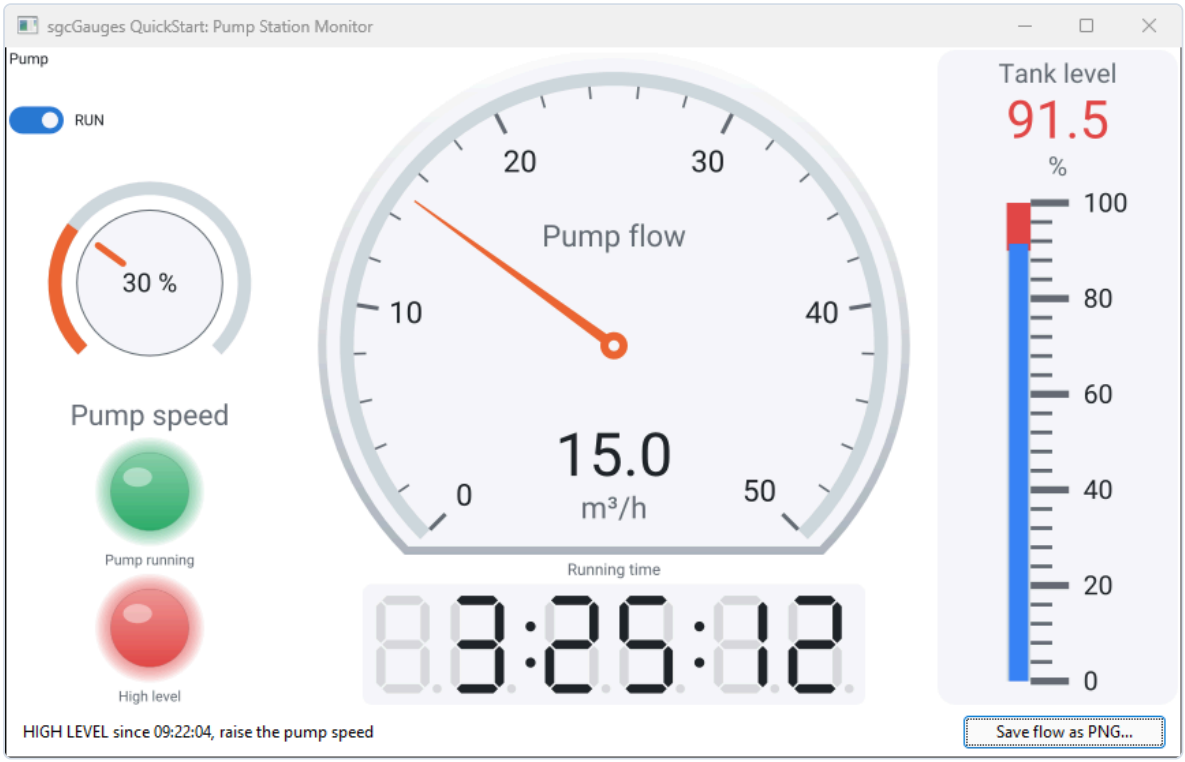


Figure 4.5. Demo 05 about a second after the start. The tank level began at 89 % and has just passed 90 %, so the High level LED blinks and the status line reports the alarm.

Components on the form

Component	Class	Properties you set
`swRun`	`TsgcVCLToggleSwitchGauge`	`Title = 'Pump'`, `Checked = True`, `OnText = 'RUN'`, `OffText = 'STOP'`, `OnChange = swRunChange`.
`knbSpeed`	`TsgcVCLKnobGauge`	`Title = 'Pump speed'`, `Units = '%'`, `ValueFormat = '0'`, `Step = 5`, `Pointers[0].Value = 30`.
`ledPump`, `ledHigh`	`TsgcVCLLEDGauge`	`Title = 'Pump running'` with a green `OnColor`; `Title = 'High level'` with a red `OnColor` and `Blink = True`.
`gFlow`	`TsgcVCLRadialGauge`	`Scales[0]`: `Maximum = 50`, `MajorInterval = 10`. `Title = 'Pump flow'`, `Units = 'm³/h'`, `ValueFormat = '0.0'`.
`gTank`	`TsgcVCLLinearGauge`	Vertical. `Pointers[0]`: `Kind = gpkBar`, blue. A red range from 90 to 100. `Thresholds[0]`: `Value = 90`, `Direction = gtdBoth`, `Hysteresis = 5`, red `ValueColor`. `OnThreshold = gTankThreshold`.

Component	Class	Properties you set
`gRuntime`	`TsgcVCLDigitalGauge`	`Scales[0].Maximum = 359999`, `Digits = 6`, `Title = 'Running time'`, `OnFormatValue = gRuntimeFormatValue`.
`tmrSim`	`TTimer`	`Interval = 200`, `OnTimer = tmrSimTimer`.
`btnSnapshot`, `dlgSave`	`TButton`, `TSaveDialog`	`OnClick = btnSnapshotClick`; the dialog has `DefaultExt = 'png'` and a PNG filter.
`lblStatus`	`TLabel`	The status line at the bottom.

The input gauges are configured like any other gauge. The knob keeps its value in `Pointers[0]`, and the switch has `Checked`:

```
object swRun: TsgcVCLToggleSwitchGauge
  Title = 'Pump'
  Checked = True
  OnText = 'RUN'
  OffText = 'STOP'
  Align = alTop
  TabOrder = 0
  OnChange = swRunChange
end
object knbSpeed: TsgcVCLKnobGauge
  Pointers = <
    item
      Value = 30.000000000000000000
    end>
  Title = 'Pump speed'
  Units = '%'
  ValueFormat = '0'
  Step = 5.000000000000000000
  Align = alTop
  TabOrder = 1
end
```

The code

`FormCreate` sets the starting state of the simulation. The LED state is set here because `Active` of an LED is not stored in the form file:

```
const
  CS_MAX_FLOW = 50; // m3/h of the pump at 100 %

procedure TfrmGaugeMonitorApp.FormCreate(Sender: TObject);
begin
  // the starting state of the simulation
  FLevel := 89;
  FFlow := CS_MAX_FLOW * knbSpeed.Value / 100;
  FRuntimeMs := 12310 * 1000; // 3:25:10 of running time so far
  gTank.Value := FLevel;
  gFlow.Value := FFlow;
  gRuntime.Value := FRuntimeMs div 1000;
  // Active of a LED is not stored in the DFM, it is always set in code
  ledPump.Active := swRun.Checked;
  lblStatus.Caption := 'Level normal';
end;

procedure TfrmGaugeMonitorApp.swRunChange(Sender: TObject);
begin
  ledPump.Active := swRun.Checked;
end;
```

The timer is the simulation loop. It reads the input gauges and writes the output gauges:

```
procedure TfrmGaugeMonitorApp.tmrSimTimer(Sender: TObject);
var
  vInflow, vTarget: Double;
begin
  Inc(FStep);
  // the inflow changes slowly between 14 and 22 m3/h
  vInflow := 18 + 4 * Sin(FStep * 0.02);
  // the pump follows the knob with a short delay; a stopped pump gives 0
  if swRun.Checked then
    vTarget := CS_MAX_FLOW * knbSpeed.Value / 100
  else
    vTarget := 0;
  FFlow := FFlow + (vTarget - FFlow) * 0.2;
  // the level rises when more water comes in than the pump takes out
  FLevel := FLevel + (vInflow - FFlow) * 0.05;
  if FLevel < 0 then
    FLevel := 0;
  if FLevel > 100 then
    FLevel := 100;
  if swRun.Checked then
    Inc(FRuntimeMs, Integer(tmrSim.Interval));
  // write the new readings: the gauges animate and check the threshold
  gFlow.Value := FFlow;
  gTank.Value := FLevel;
  gRuntime.Value := FRuntimeMs div 1000;
end;
```

The alarm handler, the custom text of the running time and the snapshot:

```
procedure TfrmGaugeMonitorApp.gTankThreshold(Sender: TObject;
  aThreshold: TsgcGaugeThreshold; aValue: Double; aRising: Boolean);
begin
  ledHigh.Active := aThreshold.Active;
  if aThreshold.Active then
    lblStatus.Caption := 'HIGH LEVEL since ' + FormatDateTime('hh:nn:ss',
      Now) + ', raise the pump speed'
  else
    lblStatus.Caption := 'Level normal again at ' +
      FormatDateTime('hh:nn:ss', Now);
end;

procedure TfrmGaugeMonitorApp.gRuntimeFormatValue(Sender: TObject;
  aPointer: Integer; aValue: Double; var aText: string);
var
  vSeconds: Integer;
begin
  { aText already holds the text made with ValueFormat; replace it. The
    seven segment display draws ':' as a narrow colon cell. }
  vSeconds := Round(aValue);
  aText := Format('%d:%.2d:%.2d', [vSeconds div 3600, (vSeconds div 60) mod
    60, vSeconds mod 60]);
end;

procedure TfrmGaugeMonitorApp.btnSnapshotClick(Sender: TObject);
begin
  // SaveToPNG renders the gauge at its client size; 2 doubles the pixels
  if dlgSave.Execute then
    gFlow.SaveToPNG(dlgSave.FileName, 2);
end;
```

How it works

1. **Inputs.** The knob and the switch are read like properties: `knbSpeed.Value` and `swRun.Checked`. Only the switch has an `OnChange` handler, to light the pump LED at once. `OnChange` fires only for changes made by the user, so `FormCreate` sets the LED itself.
2. **The loop.** Every 200 milliseconds the timer computes a new flow and a new level and writes them, together with the running time, to the output gauges. Each write animates the gauge and checks its thresholds.
3. **The alarm.** The level starts at 89 %, and with the knob at 30 % the pump takes out less water than flows in, so the level passes 90 % after about a second. The threshold fires `gTankThreshold`, which lights the blinking LED and updates the status line. Turn the knob to 60 % or more and the tank drains; below 85 % (90 minus the hysteresis of 5) the handler is called again and the alarm clears.
4. **Custom text.** The running time is stored as seconds, a plain number on the scale. `OnFormatValue` turns it into `h:mm:ss`. The scale maximum of 359999 seconds is 99:59:59.
5. **Export.** `SaveToPNG` renders the gauge at its client size; the second parameter scales it, so 2 gives an image with twice the pixels in each direction. `SaveToSVG` works the same way.

Try this

- Replace the knob with a `TsgcVCLSliderGauge` named `knbSpeed`. It has the same `Value`, `Step` and `OnChange`, so the code does not change.
- Add a `TsgcVCLPushButtonGauge` that resets the running time while it is pressed.
- Call `gTank.ShowPrintPreview` or `gTank.Print`; `PrintSettings` holds the page settings.

5. Main components

The SGC Gauges palette page holds one control per gauge type, four input gauges and one non visual component. This chapter lists them, then groups them by what they are for, and finally introduces the model classes you meet in code.

Every class has its own help topic. The control topics are in the Components section of the sgcGauges help, one per gauge type, and list the properties, methods and events with their types and defaults.

Reference table

Class	Palette page	What it is for	Key members
`TsgcVCLRadialGauge`	SGC Gauges	A dial with one or more needles	`RadialStyle`, `StartAngle`, `SweepAngle`, `ValuePosition`, `Scales`, `Pointers`, `Thresholds`
`TsgcVCLLinearGauge`	SGC Gauges	A bar, thermometer or bullet graph	`Orientation`, `LinearStyle`, `Pointers[0].Kind`
`TsgcVCLDigitalGauge`	SGC Gauges	A segment display	`Digits`, `DisplayType`, `OnFormatValue`
`TsgcVCLLEDGauge`	SGC Gauges	A status lamp	`Active`, `OnColor`, `Blink`
`TsgcVCLLEDBarGauge`	SGC Gauges	A row of LED segments	`Segments`, `Scales[0].Ranges`
`TsgcVCLProgressRingGauge`	SGC Gauges	A progress ring	`Thickness`, `RoundCaps`, `ShowPercent`
`TsgcVCLTankGauge`	SGC Gauges	A tank with a liquid level	`Value`, `Scales`, `Thresholds`
`TsgcVCLSymbolGauge`	SGC Gauges	A process symbol for HMI screens	`Kind`
`TsgcVCLKnobGauge`, `TsgcVCLSliderGauge`	SGC Gauges	Input of a value	`Value`, `Step`, `OnChange`
`TsgcVCLToggleSwitchGauge`, `TsgcVCLPushButtonGauge`	SGC Gauges	Input of on and off	`Checked`, `OnText`, `OffText`, `OnChange`
`TsgcVCLVUMeterGauge`, `TsgcVCLClockGauge`, `TsgcVCLCompassGauge`, `TsgcVCLAnnunciatorGauge`	SGC Gauges	VU meter, clock, compass, alarm annunciator (not in Community)	See their help topics
`TsgcGaugeDataLink`	SGC Gauges	Binds a gauge to a dataset field	`Gauge`, `DataSource`, `DataField`
`TsgcFMX...Gauge`	SGC Gauges (FireMonkey)	The FireMonkey versions of all the controls above	The same as the VCL controls

Every VCL gauge descends from `TsgcVCLCustomGauge`, which publishes the shared members: `Theme`, `Skin`, `SkinData`, `Animation`, `Title`, `Units`, `ValueFormat`, `ShowValue`, `OnThreshold` and `PrintSettings`, and offers `Value`, `Gauge`, `SaveToPNG`, `SaveToSVG`, `Print` and `ShowPrintPreview`.

Display gauges

Radial and linear gauges

The two workhorses. The radial gauge is the classic dial; `RadialStyle` chooses a full dial, a half dial, a quarter dial or a custom arc set by `StartAngle` and `SweepAngle`. The linear gauge is a straight scale, horizontal or vertical, drawn as a bar, a thermometer or a bullet graph. Both can hold several scales and pointers (Registered edition), for example a speedometer with an outer km/h scale and an inner mph scale.

Digital gauge, LED and LED bar

The digital gauge shows the value in seven segment digits, like an odometer or a panel meter; `Digits` sets the number of cells, and `OnFormatValue` lets you show text such as a time. The LED is a single lamp, switched with `Active`, with an optional blink. The LED bar is a row of segments that light up to the value and take their colours from the ranges of the scale.

Progress ring, tank and symbol

The progress ring is a ring that closes as the value grows, good for percentages. The tank shows a liquid level, and the symbol gauge draws process symbols such as pumps and valves. The tank and the symbol are made for HMI style panels; chapter 7 points to the `HMIPanel` demo.

Input gauges

The knob and the slider set a number; the toggle switch and the push button set on or off. They follow the theme and the skin of the display gauges, so an input panel looks like part of the instrument. They react to the mouse, the keyboard (arrows, PageUp, PageDown, Home, End, Space), touch and the mouse wheel, and their `OnChange` fires only for user changes. The push button is momentary by default and can be made latching with its `Mode` property.

Data and live feeds

`TsgcGaugeDataLink` binds a gauge to a field of a dataset: set `Gauge`, `DataSource` and `DataField`, and the gauge shows the field value of the current record. In the Registered edition a user change on an input gauge is written back to the field.

The sgcGaugesWS bridge, installed with sgcWebSockets, adds three feed sources: `TsgcGaugeFeedSource_WebSocket`, `TsgcGaugeFeedSource_MQTT` and `TsgcGaugeFeedSource_SSE`. They bind gauges to live messages without code. Without the bridge, write `Value` from your own code, which is safe from a background thread.

The model classes

Each control owns a model of the matching type, returned by its `Gauge` property: `TsgcRadialGauge`, `TsgcLinearGauge`, `TsgcDigitalGauge`, `TsgcLEDGauge`, `TsgcLEDBarGauge`, `TsgcProgressRingGauge`, `TsgcTankGauge`, `TsgcSymbolGauge`, and for input `TsgcKnobGauge`, `TsgcSliderGauge`, `TsgcToggleSwitchGauge` and `TsgcPushButtonGauge`. All descend from `TsgcCustomGauge` in `sgcGauges_Model`, which also declares the collection items:

Class	What it is	Key members
`TsgcCustomGauge`	The base model	`Value`, `SetValue`, `ValueToText`, `BeginUpdate`, `EndUpdate`, `FormatSettings`
`TsgcGaugeScale`	An item of `Scales`	`Minimum`, `Maximum`, `MajorInterval`, `MinorCount`, `LabelFormat`, `Reversed`, `Ranges`
`TsgcGaugeRange`	An item of `Ranges`	`StartValue`, `EndValue`, `Color`
`TsgcGaugePointer`	An item of `Pointers`	`Value`, `Kind`, `Color`, `MarkerShape`
`TsgcGaugeThreshold`	An item of `Thresholds`	`Value`, `Direction`, `Hysteresis`, `Color`, `ValueColor`, `Enabled`, `Active`
`TsgcGaugeSkin`	A skin, unit `sgcGauges_Skin`	Used through `Skin`, `SkinData` and `sgcGaugesRegisterSkin`

Because the models need no window, a console program or a server can create one, set its values and write it to PNG or SVG. The `Headless` demo shows how.

6. Common problems

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

Values and pointers

The value stops at 100

What you see. You write 150 or 48213 to `Value`, and the gauge shows 100.

Why. A value is clamped to the range of its scale when it is written, and the default scale is 0 to 100. A later change of the range does not bring back a value that was cut.

Fix. Set `Scales[0].Minimum` and `Scales[0].Maximum` first, then write the value. This applies to the digital gauge too: the odometer of demo 04 has `Maximum = 999999`.

The value set in the Object Inspector is not in the DFM

What you see. You look for `Value = 65` in the form file and do not find it, or code that reads the DFM does not see it.

Why. `Value` is a shortcut for `Pointers[0].Value` and is not stored itself. The same is true for `Active` of an LED.

Fix. Nothing is lost: the starting value is in `Pointers[0].Value`. Set it there, or in `Value`, in the Object Inspector. Set an LED `Active` in code, as demo 05 does in `FormCreate`.

The thermometer shows a thin line instead of a column

What you see. A linear gauge with `LinearStyle = glsThermometer` or `glsBar` draws the value as a thin needle.

Why. The default pointer kind is `gpkNeedle`, whatever the style of the gauge.

Fix. Set `Pointers[0].Kind` to `gpkBar` for a filled column, as demos 02, 04 and 05 do.

The second scale or pointer is not drawn

What you see. You added a second pointer or scale, but only the first one appears, and its thresholds never fire.

Why. The Community edition draws, animates and checks thresholds only for the first scale and the first pointer. The others are kept in the collections.

Fix. Use one gauge per value in Community, or use the Registered edition.

The needle jumps instead of moving

Why. `Animation.Enabled` is `False` or `Animation.Duration` is 0, or the value came from a path without animation: the designer, the form file, `SetValue` with `aAnimate = False`, or a headless render.

Fix. Check the `Animation` properties of the gauge.

Alarms and events

OnThreshold fires when the alarm starts but never when it clears

Why. With the default `Direction = gtdRising`, and with `gtdFalling`, the event fires only when the threshold becomes active.

Fix. Set `Direction` to `gtdBoth` and read `aRising`, or `aThreshold.Active`, in the handler, as demos 03 and 05 do.

OnThreshold does not fire at all

Why. After an event the value must leave the band given by `Hysteresis` before the threshold can fire again. A change of the threshold itself, such as a new `Value`, updates `Active` without an event. The threshold may also be disabled, or watch another pointer through `PointerIndex`.

Fix. Check `Enabled`, `PointerIndex` and `Hysteresis`, and remember that only new values written to the gauge are checked.

No threshold mark on the scale

Why. A threshold draws a mark only when its `Color` is not 0. In demo 05 the tank threshold has only a `ValueColor`, so it has no mark.

The input gauge OnChange does not fire when my code sets the value

Why. That is by design: `OnChange` of the knob, slider, switch and button fires only for user changes, so a handler that writes the value cannot start a loop.

Fix. After setting the value in code, run the same update yourself, as demo 05 does for the pump LED in `FormCreate`.

Look and text

The gauges show a point on a machine that uses a decimal comma

Why. The gauges use their own format settings, with a point, so their text does not depend on the machine.

Fix. Assign `Gauge.FormatSettings`, as shown in chapter 3.

A skin does not change the colours I set

Why. A colour set on the gauge itself, such as `FaceColor` or `Pointers[0].Color`, always wins over the skin and the theme. Under the high contrast theme no skin is applied at all.

Fix. Set the colour property back to 0 to let the skin decide.

The text is too small or too large

Why. Font sizes are relative to the size of the gauge, so the text follows the control.

Fix. Change the `Size` of `TitleFont`, `ValueFont` or the `LabelFont` of a scale to change the proportion.

Editions and compiling

The compiler cannot find sgcGauges_VCL

Why. The library path of your Delphi version does not contain `sgcUI\Source`, for example because the packages were built by hand or the project uses a different platform.

Fix. Add the folder in **Tools** > **Options** > **Language** > **Delphi Options** > **Library**, for the platform you compile for, Win32 or Win64.

TsgcVCLClockGauge or another class is not declared

Why. The VU meter, the clock, the compass and the annunciator are part of the Registered edition. In Community their controls are not registered.

Fix. Code that must build in both editions can test the define `SGC_GAUGES_ADVANCED` after `{$I sgcVer.inc}`, as the Editions help topic shows.

A 14 segment display shows 7 segments

Why. The 14 segment and dot matrix displays are part of the Registered edition. In Community they draw as 7 segment.

Values written from a worker thread

`Value`, `SetValue` and the pointer `Value` may be written from any thread, and the controls repaint and call `OnThreshold` in the main thread. Every other property, the scale and the colours included, must be changed in the main thread.

7. Where next

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

The other Delphi demos

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

Demo	What it shows
`Gallery`	Every sgcGauges control for the VCL on one form, with a light and dark theme switch, an animation switch, a value simulation and PNG export. The natural next step after demo 05.
`GalleryFMX`	Every sgcGauges control for FireMonkey in a scroll box that flows the gauges to the width of the window, for Windows and Android.
`HMIPanel`	Two tanks in series with their levels, discharge pressures, flows and pumps, and an annunciator that raises and clears its alarms from `OnThreshold`.
`DashboardGauges`	Six gauges inside the tiles of a `TsgcVCLDashboard`, with the theme set on the dashboard and passed on by every tile to its gauge.
`Headless`	A console program with no VCL or FireMonkey unit: it builds one gauge model of each type in code and writes a PNG and an SVG file of each, on Windows and Linux64.
`MQTTLive`	A `TsgcGaugeFeedSource_MQTT` of the sgcGaugesWS bridge binds six gauges to `sensors/#` topics, with a publish simulator for a local broker.

Some of these demos use features or products that are not part of every edition, such as the annunciator, sgcDashboard or the sgcWebSockets bridge. Their readme files say what they need.

Help topics

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

Topic	What it covers
Introduction, Gauge types	What sgcGauges does, and an overview of every gauge type.
Quick Start VCL	A first gauge in a few steps, the short version of this guide.
Ranges, Pointers, Animation	The building blocks in depth.
Thresholds	Directions, hysteresis and `OnThreshold`, with a high and low alarm example.
Themes	Themes, colours and the number format.
Printing, Headless rendering	Print, preview, PNG and SVG output.
Editions	What each edition includes, and code that builds in all of them.
Bridge overview	Live feeds from WebSocket, MQTT and SSE through sgcWebSockets.
One topic per control	The radial, linear, digital, LED, tank and the other gauges, with every property, its type and its default.

On the website

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

Enjoy building your instrument panels.

---

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