sgcEditors

Getting Started

Date pickers, spin edits, ratings and validated data entry forms 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 better editors on it: a date picker with a real calendar, a spin edit that knows its units, a currency edit, a star rating, a colour picker, input that checks itself, and a data entry form that builds itself from a dataset. You do not need to know anything about sgcEditors in advance.

The guide follows the five QuickStart demos. Each demo adds one idea to the previous one, from four editors set up without a single line of code to a small contacts application with a list, a generated data form, validation and a navigator. 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, type into the editors, then read its section and try the suggested changes. Chapters 1 to 3 explain what sgcEditors 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\Editors\Delphi\00.QuickStart` folder of your sgcUI installation. Each demo has its own subfolder, from `01.HelloEditors` to `05.EditorContactsApp`, with a Delphi project (`.dpr`) that you can open and run directly. All of them are VCL applications for Windows. Demos 01 to 04 use only features that are part of every edition; demo 05 uses the data form, which is part of the Registered and Trial editions.

The screenshots were taken on a Windows machine with Spanish regional settings, so they show `1.249,99 $` and `25/10/2026`. The editors format numbers, currency amounts and dates with the Windows regional settings of the machine that runs the demo, so on your machine the same values may look like `$1,249.99` and `10/25/2026`.

Contents

1. What sgcEditors is

The idea in one minute
Your first form needs no code
What you need
Part of sgcUI
Coming from the standard VCL editors

2. Install and the palette

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

3. Core concepts

One control, one typed value
OnChange, for the user and for your code
Limits come before values
Creating an editor in code
Theme and accent colour
Validation lives in sgcUICore
Adding a rule in code
Data aware editors and the data form
Regional settings
Summary: who does what

4. The QuickStart demos

4.1 Demo 01: Hello Editors
4.2 Demo 02: Editor Values
4.3 Demo 03: Editor Themes
4.4 Demo 04: Editor Validation
4.5 Demo 05: Editor Contacts App

5. Main components

Reference table
Dates and times
Numbers and ratings
Choices, colours and tokens
Text editors
The validator and the summary
Layout control, data form and data aware editors

6. Common problems

Values and input
Validation
The data form
Editions and compiling

7. Where next

The other Delphi demos
Help topics
On the website
Getting help

1. What sgcEditors is

sgcEditors is a set of input controls for Delphi, for the VCL and for FireMonkey. It covers the fields that every business form needs and that the standard controls handle only partly: a month calendar, date, time and date and time pickers, a spin edit, a range slider, a star rating, check and radio groups, a colour picker, a token edit, a button edit, a search edit, a masked edit, a currency edit, a lookup edit, a memo and a numeric keypad for touch screens.

Next to the editors there are three bigger pieces: data aware versions of the editors for a `TDataSource`, a layout control that arranges fields in groups and tabs, and a data form that builds a complete data entry form from a dataset. The validator that checks a form, with its red borders, icons and summary list, comes from sgcUICore and works with the sgcEditors controls and with the standard VCL controls alike.

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

The idea in one minute

Think of an order line on a form: a delivery date, a quantity, a unit price and a rating of the service. With the standard VCL you would use a `TDateTimePicker`, a `TSpinEdit` from the Samples page, a plain `TEdit` for the price with your own code to parse and format it, and probably some images for the stars.

With sgcEditors each of these fields is one control with one typed value:

Field	Control	Its value
Delivery date	`TsgcVCLDatePicker`	`Date: TDate`, with a drop down calendar
Quantity	`TsgcVCLSpinEdit`	`Value: Double`, between `MinValue` and `MaxValue`, with a unit such as `pcs`
Unit price	`TsgcVCLCurrencyEdit`	`Value: Currency`, formatted with the currency symbol and the separators
Service rating	`TsgcVCLRating`	`Value`, the number of stars

You configure the controls in the Object Inspector, you read and write their value in code, and every control fires `OnChange` when its value changes. All of them share one base class, so the look (`Theme` and `AccentColor`) is set the same way on every editor. That is the entire idea; everything else is detail.

Your first form needs no code

The first QuickStart demo is that order line. Its unit has no event handlers at all: every value is set in the Object Inspector and stored in the form file. These are two of its four editors, as the form file stores them:

```
  object spnQuantity: TsgcVCLSpinEdit
    Left = 256
    Top = 44
    Width = 200
    Height = 28
    TabOrder = 1
    MinValue = 1
    MaxValue = 99
    Units = 'pcs'
    Value = 3
  end
  object curPrice: TsgcVCLCurrencyEdit
    Left = 24
    Top = 124
    Width = 200
    Height = 28
    TabOrder = 2
    Decimals = 2
    CurrencySymbol = '$'
    AllowNegative = False
    Value = 1249.99
  end
```

Run it and the spin edit already refuses a quantity above 99, the currency edit accepts only digits and shows the grouping separators, and the date picker opens a calendar with Alt+Down. The `uses` clause of the form names the units of the controls; Delphi adds them for you when you drop the components:

```
uses
  Classes, Controls, Forms, StdCtrls,
  // sgc
  sgcEditors_VCL, sgcEditors_VCL_Date, sgcEditors_VCL_Value,
  sgcEditors_VCL_Mask;
```

What you need

- **Delphi.** Delphi 7, 2007, 2009, 2010, XE to XE8, 10 Seattle to 10.4 Sydney, 11, 12 and 13. Every version has its own set of packages.
- **VCL.** The `TsgcVCL` controls run on Windows, Win32 on every Delphi version and Win64 from Delphi XE2.
- **FireMonkey.** The `TsgcFMX` twins of the controls exist from Delphi XE6 on, with the same properties and events. This guide uses the VCL only, but what you learn applies to both.
- **sgcUICore.** The shared layer of sgcUI with the canvas, the themes and the validator. The installer installs it automatically.
- **For demo 05:** the Registered or Trial edition, because the data form is not part of the Community edition, and the `MidasLib` unit of your Delphi edition for the in-memory `TClientDataSet`.

Part of sgcUI

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

- An editor looks the same on the VCL and on FireMonkey, and it matches the other sgcUI controls on your form. The `Theme` property (`utDefault`, `utLight`, `utDark`, `utCustom`, `utHighContrast`) works the same way on all of them.
- The validator, `TsgcVCLValidator`, and its summary list, `TsgcVCLValidationSummary`, belong to sgcUICore. They check sgcEditors controls and standard VCL controls on the same form, and other sgcUI products use them too.
- Other products build on sgcEditors. The sgcGrid VCL and FireMonkey packages require the sgcEditors runtime packages, which is why the installer installs sgcEditors whenever you select sgcGrid.

Coming from the standard VCL editors

If you use `TEdit`, `TMaskEdit`, `TDateTimePicker`, `TSpinEdit` and `TRadioGroup` today, nothing forces you to replace them. The validator checks them as they are, and demo 04 mixes two plain `TEdit` fields with sgcEditors controls on one form.

The difference shows where a standard control leaves the work to you. A currency edit gives you a `Currency`, not a string to parse. A spin edit takes fractions, draws its unit and wraps around if you ask it to. A date picker can refuse dates before a minimum, after a maximum, on given weekdays or on a list of holidays. Every editor follows the light and dark themes and an accent colour, which the standard controls cannot do on older Delphi versions without VCL styles. And when the editors sit on a dataset, the data form can create them, lay them out and validate them for you.

2. Install and the palette

This chapter gets sgcEditors 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

sgcEditors 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 **sgcEditors**. The installer shows one checkbox per product and enables the products your licence includes. sgcUICore has no checkbox: it is installed automatically with sgcEditors. Selecting **sgcGrid** also installs sgcEditors, because the grid uses its editors.
4. 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 sgcEditors units without any setting on your side.

The packages

You only need this section if you build the packages yourself. sgcEditors 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
`sgcEditors<V>`	Runtime	The models of the controls, the date maths, the layout and the data form models. No VCL, no FireMonkey. Requires sgcUICore.
`sgcEditorsVCL<V>`	Runtime	The `TsgcVCL` controls. Requires `sgcUICoreVCL` and `sgcEditors`.
`sgcEditorsFMX<V>`	Runtime, XE6 and later	The `TsgcFMX` controls.
`dclsgcEditors<V>`	Design time	Registers the VCL controls on the palette, with the component editor of the data form.
`dclsgcEditorsFMX<V>`	Design time, XE6 and later	Registers the FireMonkey controls. Install `dclsgcEditors` first.

Build them in dependency order: sgcUICore first (`sgcUICore`, then `sgcUICoreVCL`), then `sgcEditors`, then `sgcEditorsVCL`, then `dclsgcEditors`. 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 Editors**. In a VCL project it shows these components:

Group	Components
Dates and times	`TsgcVCLMonthCalendar`, `TsgcVCLDatePicker`, `TsgcVCLTimePicker`, `TsgcVCLDateTimePicker`
Values	`TsgcVCLSpinEdit`, `TsgcVCLRangeSlider`, `TsgcVCLRating`
Choices and colours	`TsgcVCLCheckGroup`, `TsgcVCLRadioGroup`, `TsgcVCLColorPicker`, `TsgcVCLTokenEdit`
Text	`TsgcVCLButtonEdit`, `TsgcVCLSearchEdit`, `TsgcVCLMaskEdit`, `TsgcVCLCurrencyEdit`, `TsgcVCLLookupEdit`, `TsgcVCLMemo`
Touch	`TsgcVCLNumPad`, `TsgcVCLNumPadPopup`
Layout	`TsgcVCLLayoutControl`
Data aware (Registered and Trial)	`TsgcVCLDBDatePicker`, `TsgcVCLDBTimePicker`, `TsgcVCLDBSpinEdit`, `TsgcVCLDBRating`, `TsgcVCLDBCheckGroup`, `TsgcVCLDBRadioGroup`, `TsgcVCLDBRangeSlider`, `TsgcVCLDBButtonEdit`, `TsgcVCLDBMaskEdit`, `TsgcVCLDBCurrencyEdit`, `TsgcVCLDBMemo`, `TsgcVCLDBLookupEdit`
Data form (Registered and Trial)	`TsgcVCLDataForm`

In a FireMonkey project the same page shows the `TsgcFMX` twins instead; from Delphi XE2 on, each framework sees only its own controls. The validator and the validation summary are not on this page: they belong to sgcUICore, so `TsgcVCLValidator` and `TsgcVCLValidationSummary` are on the **SGC UICore** page, which the installer adds together with sgcEditors.

Help at design time

- **Collections in the Object Inspector.** The items of a check group or a radio group, the columns of a lookup edit, the rules of a validator and the fields of a data form are collections. Click the ellipsis button of `Items`, `Columns`, `Rules` or `Fields` to open the collection editor, select an item and set its properties in the Object Inspector.
- **String lists.** `DisabledDates` of the date controls, `Swatches` of the colour picker, `Items` of the lookup edit and `Options` of a data form field open the standard string list editor.
- **The data form component editor.** Right click a `TsgcVCLDataForm` on the form and choose **Generate fields**: it fills `Fields` with one item per field of the dataset, which you then trim and caption.

Editions

sgcEditors is built from the same source for every edition. The Registered and Trial editions include everything. The Trial is for evaluation. The free Community edition has the controls, the layout control and the validation framework, but leaves out a few advanced features: the HSV square and the alpha bar of the colour picker, the auto complete list of the token edit, the second thumb of the range slider, the data aware editors and the data form.

Delphi Community has no design time package of sgcEditors, so there the controls are not on the palette and are created in code. The QuickStart demos are built in the form designer, so the most comfortable way to follow this guide is the Trial or a Registered edition.

> **Note.** Demos 01 to 04 use only features that exist in every edition, the validator of demo 04 included. Demo 05 uses `TsgcVCLDataForm`, which needs the Registered or Trial edition. Its source says so at the top of the unit.

Where the demos are

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

Folder	Project	What it shows
01.HelloEditors	`HelloEditors.dpr`	Four editors set up in the Object Inspector, no code
02.EditorValues	`EditorValues.dpr`	Typed values read and written in code, one shared `OnChange`
03.EditorThemes	`EditorThemes.dpr`	Light and dark themes and an accent colour on every editor at once
04.EditorValidation	`EditorValidation.dpr`	A validator with rules, error borders and icons, and a summary list
05.EditorContactsApp	`EditorContactsApp.dpr`	A contacts list and a generated data form on a `TClientDataSet`

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

Running your first demo

1. In Delphi, choose **File** > **Open Project** and open `Demos\Editors\Delphi\00.QuickStart\01.HelloEditors\HelloEditors.dpr`.
2. Press **F9**. The window opens with four filled editors.
3. Click into the quantity and turn the mouse wheel, press Alt+Down in the date picker, click the stars.
4. Back in the IDE, open the form, select `spnQuantity` and look at `MinValue`, `MaxValue`, `Units` and `Value` in the Object Inspector. Those four lines are all there is to it.

> **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

sgcEditors has many controls, but they all work the same way. Learn the ideas in this chapter once and every editor on the palette will feel familiar. The QuickStart demos show each idea in practice; this chapter explains what happens behind them.

One control, one typed value

Each editor holds its own value, and it is a typed value, not a string. You never parse the text of an sgcEditors control to find out what the user entered: you read the property.

Control	Value property	Type
`TsgcVCLDatePicker`	`Date`	`TDate`
`TsgcVCLSpinEdit`	`Value`	`Double` (whole numbers with `ValueType = esvInteger`, fractions with `esvFloat` and `Decimals`)
`TsgcVCLRangeSlider`	`Value`, and `EndValue` in range mode	`Double`
`TsgcVCLRating`	`Value`	`Double` (half stars with `Precision = erpHalf`)
`TsgcVCLCurrencyEdit`	`Value`	`Currency`
`TsgcVCLColorPicker`	`SelectedColor`	`TsgcUIColor`, an ARGB colour
`TsgcVCLRadioGroup`	`ItemIndex`	`Integer`
`TsgcVCLCheckGroup`	`Checked` of each item in `Items`	`Boolean`

Behind every control there is a model: a framework neutral object that holds the value and the options, takes the keys and the mouse, and draws itself. The VCL control is a thin host around it, and the FireMonkey control is another host around the same model. That is why the VCL and FireMonkey editors have the same properties, and why you will rarely need to touch the model yourself. It is public as `Model` if you ever do.

All VCL editors descend from `TsgcVCLCustomEditor` (unit `sgcEditors_VCL`). It publishes what every editor has in common: `Theme`, `AccentColor`, `TextStyle`, `ReadOnly` and `OnChange`. When you need to treat all the editors of a form alike, test for that class, as demo 03 does.

OnChange, for the user and for your code

An editor fires `OnChange` every time its value changes: when the user types, clicks, drags or turns the mouse wheel, and also when your code writes the value. This makes a common pattern very short. Demo 02 assigns one handler to the `OnChange` of four editors and computes a total in it. Its start values are written in code, the handler runs for each of them, and the total is right from the first moment without any extra call.

The consequence: do not write a value in its own `OnChange` handler unless you guard against the recursion, and remember that loading a record into the editors fires their handlers. The range slider also has `OnChangeComplete`, which fires when the user releases the mouse button or the key, or turns the wheel, after

the value moved, for work that should not run on every pixel of a drag.

Limits come before values

Editors with a range keep their value inside it. A spin edit clamps `Value` to `MinValue` and `MaxValue`, a currency edit with `AllowNegative = False` never goes below 0, and a date picker refuses a date before `MinDate`, after `MaxDate`, on one of the `DisabledWeekDays` or in the `DisabledDates` list. `MinValue` equal to `MaxValue` means no limit.

In code the order matters: set the limits, the value type and the decimals first, and the value last. A value written before its limits is clamped to the old limits. In the Object Inspector you do not need to think about it, because the form file stores the value after the limits.

Creating an editor in code

The demos drop every control in the form designer, but an editor is an ordinary control that you can create at run time. Owner, `Parent`, bounds, the limits, the value, and the event:

```
uses
  SysUtils, Classes, Controls, Forms, StdCtrls,
  // sgc
  sgcUICore_Theme, sgcEditors_VCL, sgcEditors_VCL_Value;

procedure TfrmSnip.CreateEditors;
begin
  spnSeats := TsgcVCLSpinEdit.Create(Self);
  spnSeats.Parent := Self;
  spnSeats.SetBounds(24, 44, 200, 28);
  { the limits first, then the value: a value outside the limits is clamped }
  spnSeats.MinValue := 1;
  spnSeats.MaxValue := 20;
  spnSeats.Units := 'seats';
  spnSeats.Value := 2;
  spnSeats.Theme := utDark;
  spnSeats.OnChange := SeatsChange;
end;

procedure TfrmSnip.SeatsChange(Sender: TObject);
begin
  Caption := Format('%d seats', [Round(spnSeats.Value)]);
end;
```

`utDark` is declared in `sgcUICore_Theme`, which is why that unit is in the `uses` clause. The editor units follow a simple naming scheme: `sgcEditors_VCL_Date` for the calendar and the pickers, `sgcEditors_VCL_Value` for the spin edit, the slider and the rating, `sgcEditors_VCL_Mask` for the masked and currency edits, and so on. When you drop a control in the designer, Delphi adds its unit for you.

Theme and accent colour

Every editor has a `Theme` (`utDefault`, `utLight`, `utDark`, `utCustom`, `utHighContrast`) and an `AccentColor`, the colour of the focus ring, the selected day, the checked boxes, the slider and the selected radio button. `AccentColor` is a `TsgcUIColor`, an ARGB value with the alpha in the high byte, so an opaque orange is

`$FFEB6834`. The value `0` means "use the accent of the theme".

Both properties belong to each editor. To change a whole form, loop over its controls and set them on every `TsgcVCLCustomEditor`, as demo 03 does. The form and the standard VCL controls do not follow an sgcUI theme on their own; sgcUICore gives you the colours of a theme with `sgcUIThemeColors` and converts one of them to a `TColor` with `sgcUIColorToVCL`, so you can paint them to match.

Validation lives in sgcUICore

The validator, `TsgcVCLValidator`, is a non visual component of sgcUICore. You drop it on a form and fill its `Rules` collection, one item per check. Each rule names a `Control` and a `Kind`:

Kind	Checks	Properties
`uvrRequired`	The field is not empty. The default kind.	
`uvrRange`	The value lies between two limits.	`MinValue`, `MaxValue` (strings)
`uvrLength`	The length of the text.	`MinLength`, `MaxLength`
`uvrRegex`	The text matches a regular expression.	`Pattern`
`uvrEmail`	The text is an e-mail address.	
`uvrCompare`	The value compared with another control or a value.	`CompareControl`
`uvrCustom`	Your own code decides.	The `OnValidateRule` event

A few more ideas complete the picture:

- **When it checks.** `Trigger` is `uvtExit` by default: a field is checked when the user leaves it. `uvtChange` checks while the user types, and `uvtManual` only when you call `Validate`. `Validate` checks every rule and returns `True` when the form is valid.
- **How errors show.** `Display = [uvdBorder, uvdIcon]`, the default, paints a red border and an error icon next to every invalid field. `FocusFirstInvalid` moves the focus to the first invalid field after `Validate`. `ClearErrors` removes the marks.
- **Which controls.** `Container` is the form by default. The rules work on sgcEditors controls and on standard controls such as `TEdit`.
- **The own checks of the editors.** With `AutoDiscover = True`, the default, the validator also asks every sgcEditors control on the form for its own errors: a masked edit with an incomplete mask, or a lookup edit whose text is not in its list, is invalid without any rule.
- **The field names.** The messages name the field with the caption of the `TLabel` whose `FocusControl` is that field, without the accelerator. That is why every QuickStart form sets `FocusControl` on its labels. A rule with an `ErrorMessage` of its own replaces `%s` in it with the field name.
- **The summary.** `TsgcVCLValidationSummary`, with its `Validator` property set, lists the errors, hides itself when there are none (`AutoHide`) and moves the focus to a field when the user clicks its line.

Adding a rule in code

Rules are normally set in the Object Inspector, as in demo 04, but they can be added at run time. `Rules.Add` returns the generic rule class; cast it to `TsgcVCLValidationRule` to reach the `Control` property:

```
uses
  SysUtils, Classes, Controls, Forms, StdCtrls,
  // sgc
  sgcUICore_Validation, sgcUICore_Validation_VCL, sgcEditors_VCL_Value;

procedure TfrmSnip.CreateRules;
var
  vRule: TsgcVCLValidationRule;
begin
  Validator := TsgcVCLValidator.Create(Self);
  vRule := TsgcVCLValidationRule(Validator.Rules.Add);
  vRule.Control := edtName;
  vRule.Kind := uvrRequired;
  vRule := TsgcVCLValidationRule(Validator.Rules.Add);
  vRule.Control := spnSeats;
  vRule.Kind := uvrRange;
  vRule.MinValue := '1';
  vRule.MaxValue := '10';
  vRule.ErrorMessage := '%s: at most 10 seats per booking.';
  if not Validator.Validate then
    Caption := Format('%d errors', [Validator.ErrorCount]);
end;
```

The rule kinds such as `uvrRange` are declared in `sgcUICore_Validation`, the VCL components in `sgcUICore_Validation_VCL`. `MinValue` and `MaxValue` are strings because the same rule checks numbers, dates and texts.

Data aware editors and the data form

The Registered and Trial editions add two ways to put the editors on a dataset:

- **Data aware editors.** `TsgcVCLDBDatePicker`, `TsgcVCLDBSpinEdit`, `TsgcVCLDBCurrencyEdit` and the others are the editors with a `DataSource` and a `DataField`, like the `TDBEdit` family. Setting the value of a data aware editor in code counts as a change by the user and puts the dataset in edit mode.
- **The data form.** `TsgcVCLDataForm` takes a `DataSource` and a `Fields` collection and builds the whole form: one caption and one data aware editor per field, arranged in groups and tabs on a layout control, the validation rules of every field, a summary of the errors and a navigator. `Kind = edfAuto`, the default, picks the editor from the type of the field: a date picker for a date, a currency edit for a currency field, a check box for a Boolean and a memo for a memo. When `Fields` is empty and the dataset opens, `AutoGenerate` fills it with one item per field.

The data form creates its editors when the dataset becomes active, not when the form is designed. If you want to adjust one of them, handle `OnEditorCreated`, which receives the field description and the new control. Demo 05 shows the data form in a small application.

Regional settings

The editors format and parse with the Windows regional settings of the machine they run on. A currency edit with `CurrencySymbol = '$'` shows the dollar sign, but where it goes and which separators surround the number come from the currency format of Windows, unless you set `PositivePattern` and `NegativePattern`

yourself. With an empty `CurrencySymbol` the edit uses the currency symbol of Windows, which is why the credit of demo 05 shows a euro sign on a Spanish machine. The date picker shows dates in the short date format of Windows and the calendar starts its weeks and names its days the same way.

Every editor also has a public `FormatSettings` property. It starts with the settings of the user locale, and assigning another `TFormatSettings` changes the day names, the formats and the separators of that one control.

Your code never sees any of this: `Value` is a `Currency` and `Date` is a `TDate` whatever the display looks like.

Summary: who does what

Part	Its job
The editor (`TsgcVCL...`)	Shows and edits one typed value, keeps it inside its limits, fires `OnChange`.
Its model (`Model`)	Holds the value and draws it, the same for VCL and FireMonkey. Rarely used directly.
`TsgcVCLCustomEditor`	The common base: `Theme`, `AccentColor`, `TextStyle`, `ReadOnly`, `OnChange`.
`TsgcVCLValidator` (sgcUICore)	Checks the rules and the own checks of the editors, marks the invalid fields.
`TsgcVCLValidationSummary` (sgcUICore)	Lists the errors of a validator.
`TsgcVCLDataForm`	Builds editors, layout, validation and a navigator for a dataset (Registered and Trial).

4. The QuickStart demos

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

Demo	Project	New ideas
01.HelloEditors	`HelloEditors.dpr`	Editors configured in the Object Inspector, labels with `FocusControl`
02.EditorValues	`EditorValues.dpr`	Typed values in code, one `OnChange` for several editors, `MinDate`
03.EditorThemes	`EditorThemes.dpr`	`Theme` and `AccentColor` on every editor, theme colours for the form, check and radio groups, the colour picker
04.EditorValidation	`EditorValidation.dpr`	The validator, its rules, the own checks of the editors, the summary list
05.EditorContactsApp	`EditorContactsApp.dpr`	A data form on a dataset, groups, required fields, a navigator

> **Note.** The demos need sgcEditors installed, so the components appear on the SGC Editors palette page and the `sgcUI\Source` folder is on the library path. Demos 01 to 04 use only features that are part of every edition; demo 05 needs the Registered or Trial edition. The amounts and dates in the screenshots follow the Spanish regional settings of the machine they were taken on, so your screen may show other separators, another position of the currency symbol and another date order.

4.1 Demo 01: Hello Editors

Folder `01.HelloEditors`, project `HelloEditors.dpr`.

What you will build

The smallest sgcEditors application: an order line with a delivery date, a quantity, a unit price and a service rating. Four editors, four labels and not a single line of code. Everything you see was set in the Object Inspector.

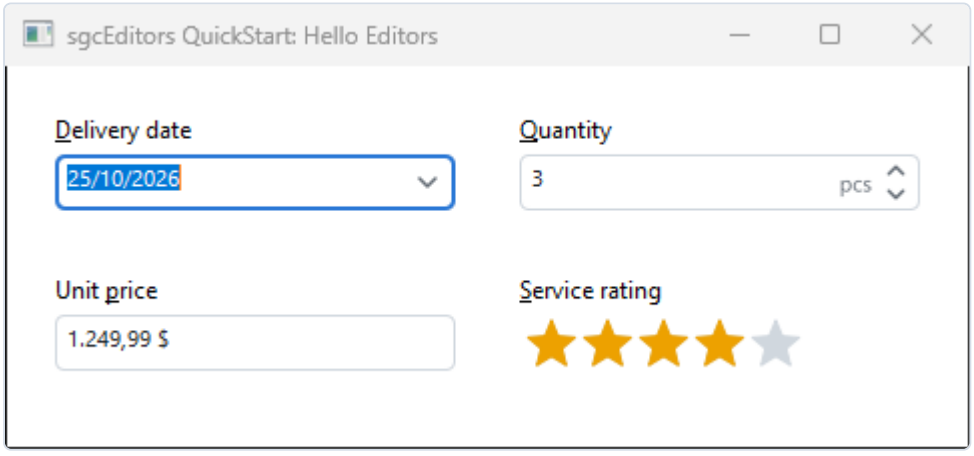


Figure 4.1. Demo 01. The date picker starts on today, the spin edit draws its unit after the number, and the currency edit formats the amount with the regional settings of Windows.

Components on the form

Component	Class	Properties you set
`dtpDelivery`	`TsgcVCLDatePicker`	None. Without a value it starts on today.
`spnQuantity`	`TsgcVCLSpinEdit`	`MinValue = 1`, `MaxValue = 99`, `Units = 'pcs'`, `Value = 3`
`curPrice`	`TsgcVCLCurrencyEdit`	`Decimals = 2`, `CurrencySymbol = '$'`, `AllowNegative = False`, `Value = 1249.99`
`rtgService`	`TsgcVCLRating`	`Value = 4`
`lblDelivery`, `lblQuantity`, `lblPrice`, `lblService`	`TLabel`	A `Caption` with an accelerator such as `'&Quantity'`, and `FocusControl` set to the editor below it

In the form file the rating and the date picker are as short as a control can be:

```
  object dtpDelivery: TsgcVCLDatePicker
    Left = 24
    Top = 44
    Width = 200
    Height = 28
    TabOrder = 0
  end
  // ...
  object rtgService: TsgcVCLRating
    Left = 256
    Top = 122
    Width = 200
    Height = 32
    TabOrder = 3
    Value = 4
  end
```

The code

There is none, apart from what Delphi writes for you. This is the whole interface section of the unit:

```
uses
  Classes, Controls, Forms, StdCtrls,
  // sgc
  sgcEditors_VCL, sgcEditors_VCL_Date, sgcEditors_VCL_Value,
  sgcEditors_VCL_Mask;

type
  TfrmHelloEditors = class(TForm)
    lblDelivery: TLabel;
    dtpDelivery: TsgcVCLDatePicker;
    lblQuantity: TLabel;
    spnQuantity: TsgcVCLSpinEdit;
    lblPrice: TLabel;
    curPrice: TsgcVCLCurrencyEdit;
    lblService: TLabel;
    rtgService: TsgcVCLRating;
  end;
```

How it works

1. The form loads from its DFM. Each editor reads its properties, the limits first and the value last, and draws itself.
2. The date picker has no stored date, so it shows today. Click the button or press Alt+Down to open its calendar, or type a date: the editor checks it as you type.
3. The spin edit keeps its value between 1 and 99. The arrows, the mouse wheel and the Up and Down keys change it, and `pcs` is drawn after the number.
4. The currency edit accepts only digits and the decimal separator, groups the thousands and adds the symbol where the regional settings of Windows put it. `AllowNegative = False` keeps the price at 0 or above.
5. The labels have `FocusControl` set, so Alt plus the underlined letter jumps to the editor. The validator of demo 04 uses the same labels to name the fields in its messages.

Try this

- Set `Count = 10` and `Precision = erpHalf` on `rtgService`. Run again: ten stars, and a click on the left half of a star gives half a star.
- Set `Increment = 5` and `Wrap = True` on `spnQuantity`. The arrows now move in steps of five and jump from 99 back to the start.
- Set `Theme = utDark` on any editor and see it change in the designer.

4.2 Demo 02: Editor Values

Folder `02.EditorValues`, project `EditorValues.dpr`.

What you will build

The order line again, now with a discount and a total. The values are written and read in code, and one event handler shared by four editors keeps the total up to date whatever the user changes.

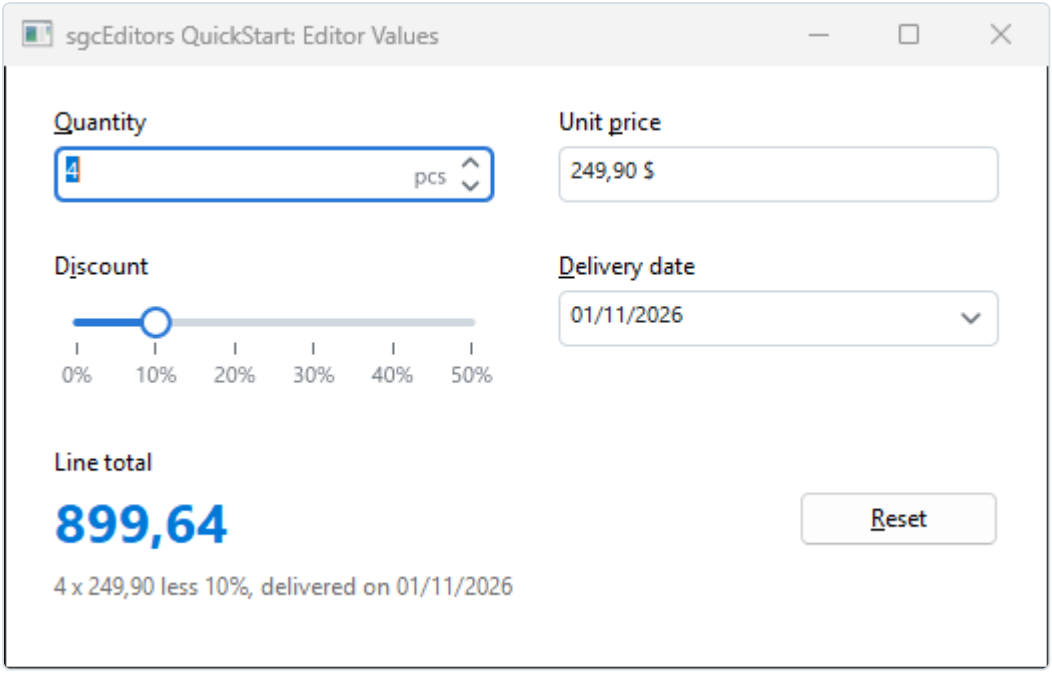


Figure 4.2. Demo 02. Four pieces at 249,90 less 10% give 899,64. Move the slider or change the quantity and the total follows at once.

Components on the form

Component	Class	Properties you set
`spnQuantity`	`TsgcVCLSpinEdit`	`MinValue = 1`, `MaxValue = 99`, `Units = 'pcs'`, `OnChange = EditorChange`
`curPrice`	`TsgcVCLCurrencyEdit`	`Decimals = 2`, `CurrencySymbol = '$'`, `AllowNegative = False`, `OnChange = EditorChange`
`sldDiscount`	`TsgcVCLRangeSlider`	`Min = 0`, `Max = 50`, `Step = 5`, `LargeStep = 10`, `TickInterval = 10`, `ShowLabels = True`, `LabelFormat = '0%'`, `OnChange = EditorChange`
`dtpDelivery`	`TsgcVCLDatePicker`	`OnChange = EditorChange`
`lblTotal`, `lblSummary`	`TLabel`	The result, a big bold total and a grey summary line
`btnReset`	`TButton`	`OnClick = btnResetClick`

The slider is the only editor with more than the usual properties. `ShowLabels` draws the scale under the track, and `LabelFormat` is a `FormatFloat` mask, so `'0%'` turns 10 into “10%”:

```
  object sldDiscount: TsgcVCLRangeSlider
    Left = 24
    Top = 112
    Width = 220
    Height = 56
    OnChange = EditorChange
    TabOrder = 2
    Min = 0
    Max = 50
    Step = 5
    LargeStep = 10
    TickInterval = 10
    ShowLabels = True
    LabelFormat = '0%'
  end
```

The code

```
procedure TfrmEditorValues.FormCreate(Sender: TObject);
begin
  { no delivery in the past: MinDate is a property like any other }
  dtpDelivery.MinDate := Date;
  DoStartValues;
end;

procedure TfrmEditorValues.DoStartValues;
begin
  { writing Value (or Date) changes what the editor shows and fires its
    OnChange, so EditorChange runs and the total follows }
  spnQuantity.Value := 4;
  curPrice.Value := 249.90;
  sldDiscount.Value := 10;
  dtpDelivery.Date := Date + 7;
end;
```

And the handler that all four editors share:

```
procedure TfrmEditorValues.EditorChange(Sender: TObject);
var
  vTotal: Currency;
begin
  { read the values back: every editor exposes a typed Value }
  vTotal := spnQuantity.Value * curPrice.Value *
    (100 - sldDiscount.Value) / 100;
  lblTotal.Caption := FormatFloat('#,##0.00', vTotal);
  lblSummary.Caption := Format('%d x %s less %d%%, delivered on %s',
    [Round(spnQuantity.Value), FormatFloat('#,##0.00', curPrice.Value),
    Round(sldDiscount.Value), DateToStr(dtpDelivery.Date)]);
end;

procedure TfrmEditorValues.btnResetClick(Sender: TObject);
begin
  DoStartValues;
end;
```

How it works

1. In the Object Inspector, on the Events tab, the `OnChange` of the four editors points to the same method, `EditorChange`.
2. `FormCreate` sets `MinDate` to today, so the calendar greys out the past and a typed date before today is refused.
3. `DoStartValues` writes four values. Each write fires `OnChange`, so `EditorChange` runs and the total is correct when the window appears.
4. `EditorChange` reads the typed values: two `Double` values, a `Currency` and a `TDate`. No string is parsed anywhere.
5. Every user change, typed, clicked, dragged or wheeled, fires `OnChange` again. **Reset** simply calls `DoStartValues`.

Try this

- Set `RangeMode = True` on the slider and read `Value` and `EndValue` as the low and the high end of a range. The second thumb needs the Registered or Trial edition; the Community edition keeps one thumb.
- Assign the slider's handler to `OnChangeComplete` instead of `OnChange`: the total is then updated when the user lets go, not during the drag.
- Set `DisabledWeekDays` on the date picker to refuse deliveries on the weekend.

4.3 Demo 03: Editor Themes

Folder `03.EditorThemes`, project `EditorThemes.dpr`.

What you will build

A hotel booking panel that changes its look. A radio group switches every editor between the light and the dark theme, and a grid of colour swatches sets the accent colour of all of them at once. The window opens dark, with an orange accent.

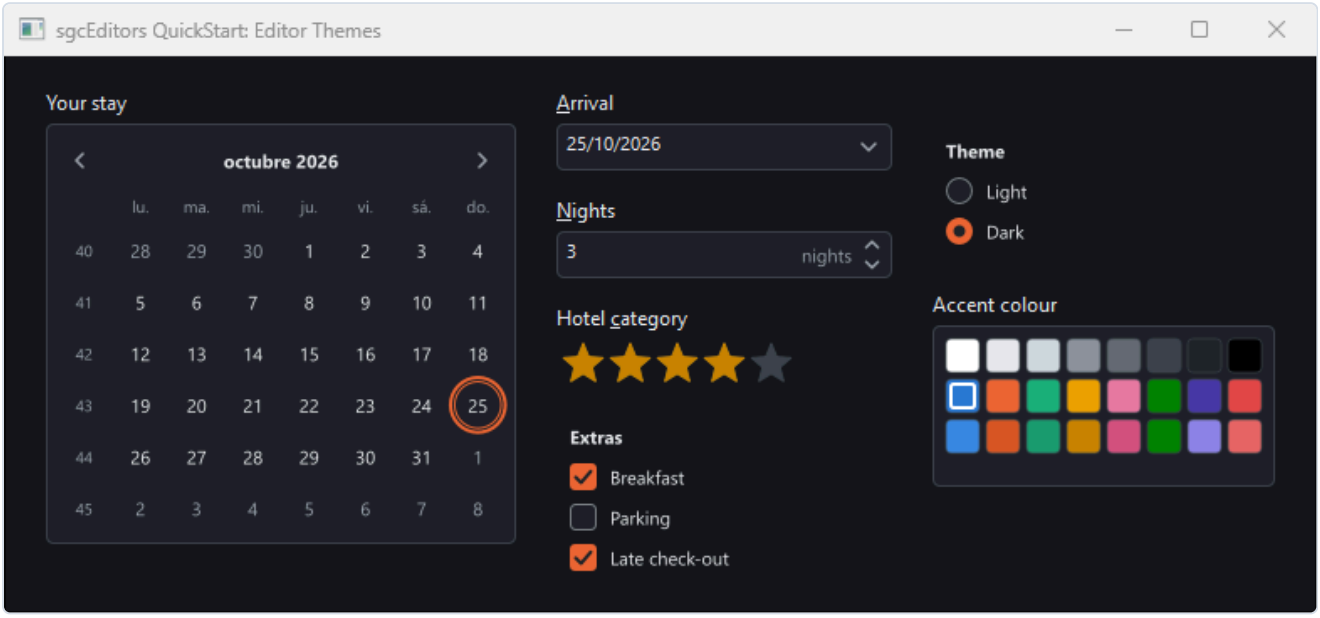


Figure 4.3. Demo 03 in the dark theme with the orange accent. The calendar, the checked boxes, the selected radio button and the stars all follow the accent colour.

Components on the form

Component	Class	Properties you set
`calStay`	`TsgcVCLMonthCalendar`	`ShowWeekNumbers = True`
`dtpArrival`	`TsgcVCLDatePicker`	None
`spnNights`	`TsgcVCLSpinEdit`	`MinValue = 1`, `MaxValue = 30`, `Units = 'nights'`, `Value = 3`
`rtgHotel`	`TsgcVCLRating`	`Value = 4`
`chkExtras`	`TsgcVCLCheckGroup`	`Caption = 'Extras'`, three `Items`, two of them `Checked`
`rdgTheme`	`TsgcVCLRadioGroup`	`Caption = 'Theme'`, `Items` Light and Dark, `ItemIndex = 1`, `OnChange = rdgThemeChange`
`clrAccent`	`TsgcVCLColorPicker`	`Parts = [ecpSwatches]`, `OnSelect = clrAccentSelect`

A check group and a radio group keep their entries in an `Items` collection. Each item has a `Caption` for the user and a `Value` for your code, and the check group items have their own `Checked`:

```
  object chkExtras: TsgcVCLCheckGroup
    Left = 328
    Top = 210
    Width = 200
    Height = 104
    TabOrder = 4
    Items = <
      item
        Caption = 'Breakfast'
        Value = 'breakfast'
        Checked = True
      end
      item
        Caption = 'Parking'
        Value = 'parking'
      end
      item
        Caption = 'Late check-out'
        Value = 'late'
        Checked = True
      end>
    Caption = 'Extras'
  end
```

The code

The `uses` clause adds three sgcUICore units: the colour type, the themes and the conversion to VCL colours.

```
uses
  Classes, Controls, Forms, StdCtrls,
  // sgc
  sgcUICore_Canvas, sgcUICore_Theme, sgcUICore_Theme_VCL, sgcEditors_VCL,
  sgcEditors_VCL_Date, sgcEditors_VCL_Value, sgcEditors_VCL_CheckGroup,
  sgcEditors_VCL_RadioGroup, sgcEditors_VCL_Color;
```

```
procedure TfrmEditorThemes.FormCreate(Sender: TObject);
begin
  { the radio group starts on Dark (ItemIndex = 1 in the DFM) and the
    accent starts orange; the user can change both afterwards }
  rdgThemeChange(nil);
  DoApplyAccent($FFEB6834);
end;

procedure TfrmEditorThemes.DoApplyTheme(aTheme: TsgcUITheme);
var
  i: Integer;
  vColors: TsgcUIThemeColors;
begin
  for i := 0 to ControlCount - 1 do
    if Controls[i] is TsgcVCLCustomEditor then
      TsgcVCLCustomEditor(Controls[i]).Theme := aTheme;
  { sgcUIThemeColors returns the colours of a theme, sgcUIColorToVCL turns
    one of them into a TColor for the standard VCL controls }
  vColors := sgcUIThemeColors(aTheme);
  Color := sgcUIColorToVCL(vColors.Background);
  Font.Color := sgcUIColorToVCL(vColors.Text);
end;
```

The accent works the same way, and the two event handlers only pass on what the user chose:

```
procedure TfrmEditorThemes.DoApplyAccent(aColor: TsgcUIColor);
var
  i: Integer;
begin
  for i := 0 to ControlCount - 1 do
    if Controls[i] is TsgcVCLCustomEditor then
      TsgcVCLCustomEditor(Controls[i]).AccentColor := aColor;
end;

procedure TfrmEditorThemes.rdgThemeChange(Sender: TObject);
begin
  if rdgTheme.ItemIndex = 1 then
    DoApplyTheme(utDark)
  else
    DoApplyTheme(utLight);
end;

procedure TfrmEditorThemes.clrAccentSelect(Sender: TObject);
begin
  { SelectedColor is the ARGB colour of the swatch the user clicked }
  DoApplyAccent(clrAccent.SelectedColor);
end;
```

How it works

1. `FormCreate` calls `rdgThemeChange` directly. The radio group starts on Dark, so every editor gets `utDark`.
2. `DoApplyTheme` walks the controls of the form. Every sgcEditors control descends from `TsgcVCLCustomEditor`, so one `is` test finds them all, the radio group and the colour picker included.
3. The form and its labels are standard VCL. `sgcUIThemeColors` returns the colours of the theme, and `sgcUIColorToVCL` turns the background and text colours into `TColor` values for `Color` and `Font.Color`. The labels have `ParentFont` set, so they follow.
4. `DoApplyAccent` sets `AccentColor` the same way. `$FFEB6834` is opaque orange: `$FF` alpha, then red, green and blue.
5. A click on a swatch fires `OnSelect` of the colour picker, and its `SelectedColor` becomes the new accent. A click on Light or Dark fires `OnChange` of the radio group.

Try this

- Add a third item, High contrast, to `rdgTheme` and apply `utHighContrast` for `ItemIndex = 2`.
- Set `Parts = [ecpSwatches, ecpHex]` on the colour picker to add a hex edit under the swatches.
- Set `Theme = utDark` on a single editor in the designer to see how one editor looks on its own.

4.4 Demo 04: Editor Validation

Folder `04.EditorValidation`, project `EditorValidation.dpr`.

What you will build

A workshop registration form that checks its input. Two standard `TEdit` fields and three sgcEditors controls are checked by one validator, the invalid fields get a red border and an error icon, and a summary on the right lists what is wrong. To show you the result straight away, the form types a few wrong values when it opens and validates once.

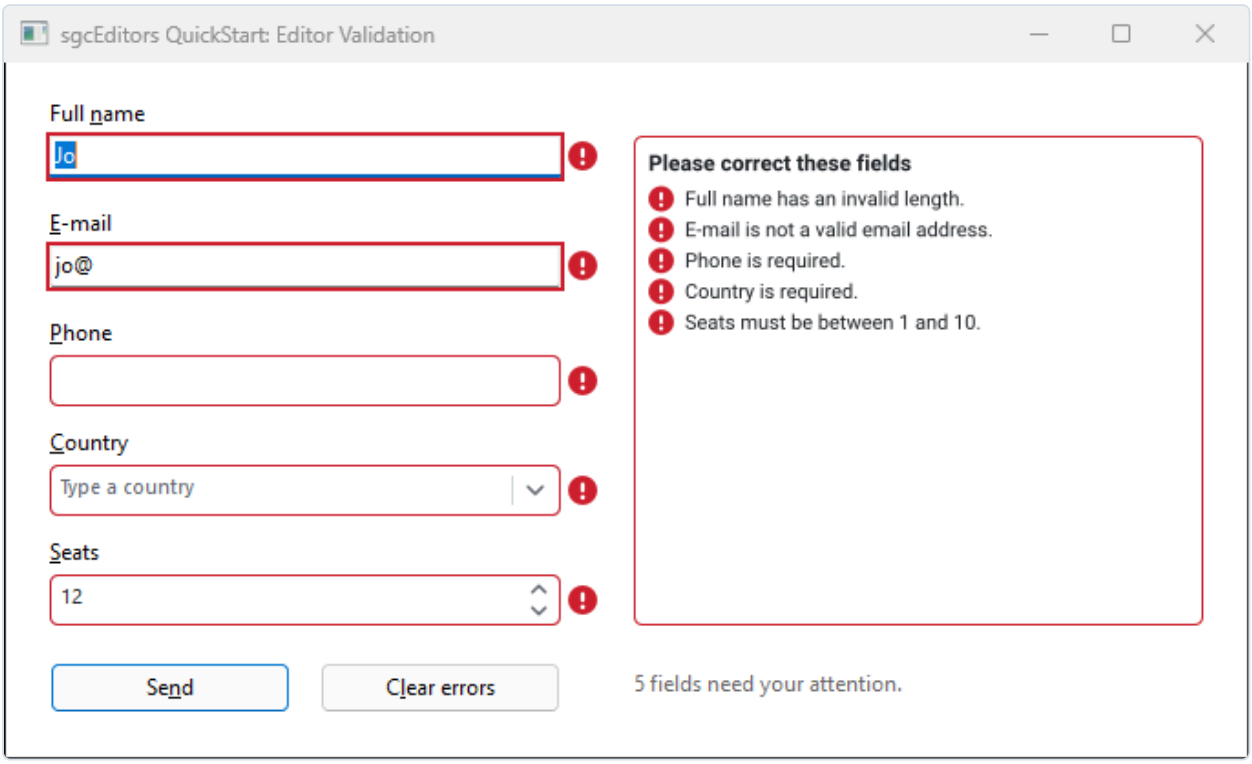


Figure 4.4. Demo 04 as it opens: a name that is too short, an e-mail without a domain, an empty phone and country, and too many seats. The status line counts the errors.

Components on the form

Component	Class	Properties you set
`edtName`, `edtEmail`	`TEdit`	None. Standard VCL edits.
`mskPhone`	`TsgcVCLMaskEdit`	`EditMask = '!\(999\) 000-0000;1;_'`
`lkpCountry`	`TsgcVCLLookupEdit`	`Items` with one `'DE|Germany'` line per country, `Columns` Code and Country, `DisplayColumn = 1`, `TextHint`
`spnSeats`	`TsgcVCLSpinEdit`	`MinValue = 1`, `MaxValue = 20`
`Validator`	`TsgcVCLValidator`	Seven `Rules`, `OnValidated = ValidatorValidated`. Everything else is left at its default.
`vsmErrors`	`TsgcVCLValidationSummary`	`Validator = Validator`, `Title = 'Please correct these fields'`
`btnSend`, `btnClear`, `lblStatus`	`TButton`, `TLabel`	`btnSend` is the `Default` button

The rules, as the form file stores them. A rule without a `Kind` line is `uvrRequired`, the default:

```
object Validator: TsgcVCLValidator
  Rules = <
    item
      Control = edtName
    end
    item
      Kind = uvrLength
      MinLength = 3
      Control = edtName
    end
    item
      Control = edtEmail
    end
    item
      Kind = uvrEmail
      Control = edtEmail
    end
    item
      Control = mskPhone
    end
    item
      Control = lkpCountry
    end
    item
      Kind = uvrRange
      MinValue = '1'
      MaxValue = '10'
      Control = spnSeats
    end>
  OnValidated = ValidatorValidated
  Left = 600
  Top = 330
end
```

The lookup edit keeps its list in `Items`, one line per entry with the columns separated by `|`. `DisplayColumn = 1` shows the country name in the edit:

```
object lkpCountry: TsgcVCLLookupEdit
  Left = 24
  Top = 220
  Width = 280
  Height = 28
  TabOrder = 3
  Items.Strings = (
    'DE|Germany'
    'ES|Spain'
    'FR|France'
    'GB|United Kingdom'
    'IT|Italy'
    'US|United States')
  Columns = <
    item
      Title = 'Code'
      Width = 44
    end
    item
      Title = 'Country'
    end>
```

```
    DisplayColumn = 1
    TextHint = 'Type a country'
  end
```

The code

```
uses
  SysUtils, Classes, Controls, Forms, StdCtrls, Dialogs,
  // sgc
  sgcUICore_Validation, sgcUICore_Validation_VCL, sgcEditors_VCL,
  sgcEditors_VCL_Value, sgcEditors_VCL_Mask, sgcEditors_VCL_Lookup;
```

```
procedure TfrmEditorValidation.FormShow(Sender: TObject);
begin
  { a name that is too short, an e-mail without a domain and too many
    seats; the phone and the country stay empty }
  edtName.Text := 'Jo';
  edtEmail.Text := 'jo@';
  spnSeats.Value := 12;
  Validator.Validate;
end;

procedure TfrmEditorValidation.btnSendClick(Sender: TObject);
begin
  { Validate checks every rule; the errors are painted on the fields, listed
    in the summary, and the first invalid field gets the focus }
  if Validator.Validate then
    ShowMessage('Thank you, ' + edtName.Text + '. Your seats are booked.');
end;

procedure TfrmEditorValidation.btnClearClick(Sender: TObject);
begin
  Validator.ClearErrors;
end;

procedure TfrmEditorValidation.ValidatorValidated(Sender: TObject;
  aValid: Boolean);
begin
  { OnValidated runs after every Validate }
  if aValid then
    lblStatus.Caption := 'The form is valid.'
  else
    lblStatus.Caption := Format('%d fields need your attention.',
      [Validator.ErrorCount]);
end;
```

How it works

1. The validator has no `Container`, so it watches its owner, the form. `Trigger` is `uvtExit`: each field is checked when the user leaves it.
2. `FormShow` writes three wrong values and calls `Validate`. The seven rules run, and five fields fail. `Display = [uvdBorder, uvdIcon]`, the default, paints the border and the icon on each of them.
3. The messages name the fields with the captions of their labels: *Full name has an invalid length.*, *E-mail is not a valid email address.*, *Phone is required.*, *Country is required.*, *Seats must be between 1 and 10.* That only works because every label has `FocusControl` set.

4. The summary is bound to the validator, so it lists the same five messages. Click a line and the focus jumps to that field. When every field is valid the summary hides itself.
5. Besides the rules, the validator asks the sgcEditors controls for their own checks: a phone number that does not fill the mask, or a country typed that is not in the list, is an error even though no rule says so.
6. `OnValidated` runs after every `Validate` and writes the error count. **Send** shows the thank you message only when `Validate` returns `True`; otherwise the first invalid field gets the focus. **Clear errors** removes the marks without changing any value.

> **Tip.** The spin edit allows up to 20 seats while the rule allows 10. The two limits do different jobs: `MaxValue` stops the user from typing a number that makes no sense at all, the rule explains a business limit with a message.

Try this

- Set `Trigger = uvtChange` on the validator. Now the e-mail turns valid the moment you type the domain.
- Give the length rule its own `ErrorMessage`, such as `'%s needs at least 3 letters.'`. The `%s` is replaced with the field name.
- Add a rule with `Kind = uvrCustom` for `edtName` and decide in `OnValidateRule` whether the value is acceptable.

4.5 Demo 05: Editor Contacts App

Folder `05.EditorContactsApp`, project `EditorContactsApp.dpr`. This demo needs the Registered or Trial edition.

What you will build

A small contacts application. A grid on the left lists the contacts, and on the right a data form edits the selected one: grouped fields, required markers, the right editor for every field and a navigator to browse, add, delete, save and cancel. The editors of demos 01 to 04 come together without being placed one by one: the data form creates them.

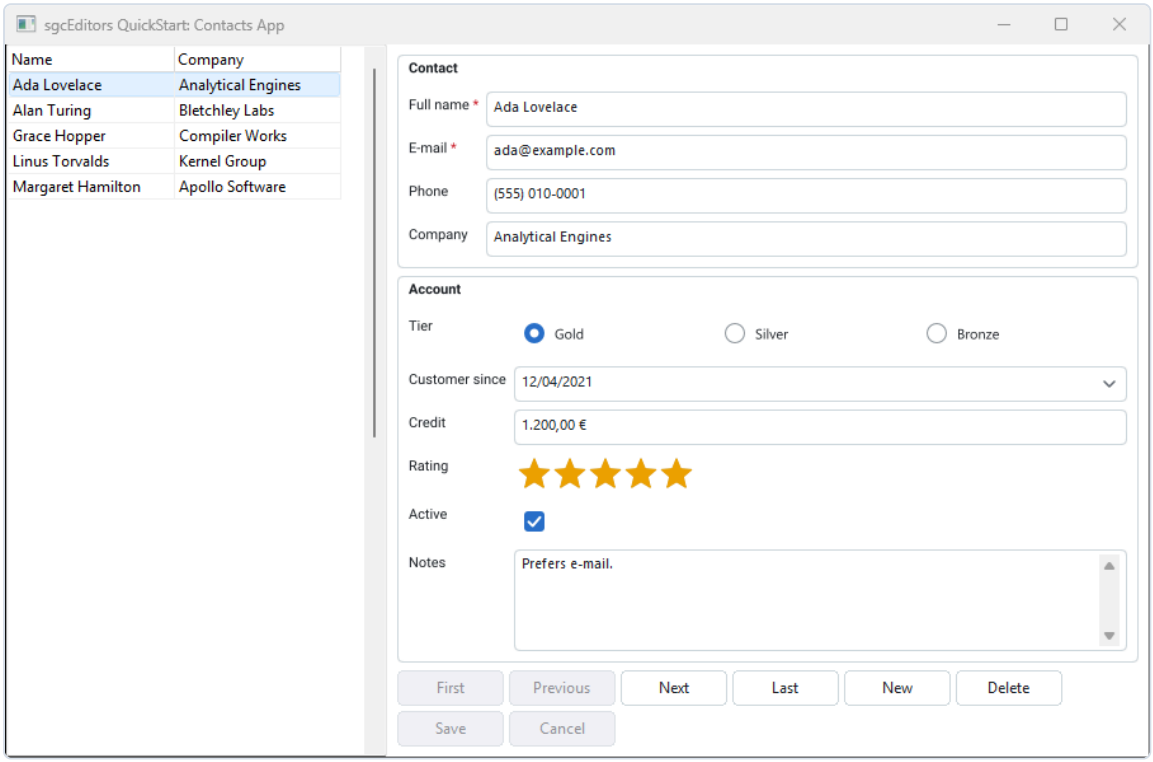


Figure 4.5. Demo 05. The two groups, the asterisks of the required fields and the navigator are all generated. Save and Cancel are disabled until a value changes.

Components on the form

Component	Class	Properties you set
`cdsContacts`	`TClientDataSet`	`FieldDefs` ID, NAME, EMAIL, PHONE, COMPANY, TIER, SINCE (date), CREDIT (currency), RATING (integer), ACTIVE (Boolean), NOTES (memo)
`dsContacts`	`TDataSource`	`DataSet = cdsContacts`
`grdContacts`	`TDBGrid`	`Align = alLeft`, `ReadOnly`, row select, columns NAME and COMPANY
`dfContact`	`TsgcVCLDataForm`	`Align = alClient`, `DataSource = dsContacts`, ten `Fields`, `NavigatorActions`

Each item of `Fields` describes one field of the form: the dataset field, the caption, the editor kind, the checks and the group. ID is not in the list, so it is not shown. An excerpt:

```
object dfContact: TsgcVCLDataForm
  Left = 300
  Top = 0
```

```
    Width = 600
    Height = 560
    Align = alClient
    TabOrder = 1
    DataSource = dsContacts
    Fields = <
      item
        FieldName = 'NAME'
        Caption = 'Full name'
        Required = True
        Group = 'Contact'
      end
      item
        FieldName = 'EMAIL'
        Caption = 'E-mail'
        Kind = edfEmail
        Required = True
        HelpText = 'name@company.com'
        Group = 'Contact'
      end
      item
        FieldName = 'PHONE'
        Caption = 'Phone'
        EditMask = '!\(999\) 000-0000;1;_'
        Group = 'Contact'
      end
      // ...
      item
        FieldName = 'TIER'
        Caption = 'Tier'
        Kind = edfChoice
        Options.Strings = (
          'Gold'
          'Silver'
          'Bronze')
        Group = 'Account'
      end
      // ...
    NavigatorActions = [edaFirst, edaPrior, edaNext, edaLast, edaNew, edaDelete, edaPost, edaCancel]
  end
```

The other fields of the Account group (SINCE, CREDIT, RATING, ACTIVE, NOTES) follow the same pattern. CREDIT has `MinValue = '0'`, RATING has `Kind = edfRating` and NOTES has `Kind = edfMemo`; SINCE and ACTIVE need nothing more than a caption, because `edfAuto` picks a date picker and a check box from their field types.

The code

The project file links `MidasLib`, so the `TClientDataSet` needs no `midas.dll`:

```
uses
  Forms,
  MidasLib,
  uEditorContactsApp in 'uEditorContactsApp.pas' {frmEditorContactsApp};
```

The unit uses the layout and data form units, the model and the VCL control of each:

```
uses
  Windows, SysUtils, Classes, Controls, Forms, DBGrids, DB, DBClient,
  // sgc
  sgcEditors_Layout, sgcEditors_VCL_Layout, sgcEditors_DataForm,
  sgcEditors_VCL_DataForm;
```

```
procedure TfrmEditorContactsApp.FormCreate(Sender: TObject);
begin
  { CreateDataSet builds the empty table from the FieldDefs of the DFM and
    opens it; the data form sees the dataset open and creates its editors }
  cdsContacts.CreateDataSet;
  { ID, NAME, EMAIL, PHONE, COMPANY, TIER, SINCE, CREDIT, RATING, ACTIVE,
    NOTES }
  cdsContacts.AppendRecord([1, 'Ada Lovelace', 'ada@example.com',
    '(555) 010-0001', 'Analytical Engines', 'Gold', EncodeDate(2021, 4, 12),
    1200, 5, True, 'Prefers e-mail.']);
  // ...
  cdsContacts.First;
end;
```

How it works

1. `CreateDataSet` builds the table from the `FieldDefs` of the form file and opens it. The data form notices that its dataset became active and builds its editors from `Fields`. Because `Fields` is not empty, nothing is generated automatically.
2. For each field it creates a caption and an editor and places them in the group named by `Group`. `Kind` chooses the editor: `edfEmail` a text edit with an e-mail check, `edfChoice` a radio group with the `Options` (up to five options; a longer list becomes a lookup edit), `edfRating` the stars, `edfMemo` a memo, and `edfAuto` whatever suits the field type.
3. `Required = True` adds an asterisk to the caption and a required rule to the validator of the form; `edfEmail` adds an e-mail rule, `MinValue` a range rule. `HelpText` becomes the hint of the editor and the grey text hint of an empty edit.
4. The grid and the data form share `dsContacts`, so a click on a row shows that contact in the form. Changing a value puts the dataset in edit mode and enables **Save** and **Cancel** in the navigator.
5. **Save** validates every field first and posts only a valid record; if a field fails, the summary of the data form lists the errors above the fields. The navigator buttons come from `NavigatorActions`; `edaPrior` and `edaPost` are captioned Previous and Save.

> **Note.** The credit in figure 4.5 shows a euro sign because the CREDIT field has no currency symbol of its own: the generated currency edit uses the currency symbol of Windows. See Regional settings.

Try this

- Set `Tab = 'Details'` on the RATING, ACTIVE and NOTES fields. The data form puts them on a tab of their own.
- Handle `OnValidateField` to refuse a credit above 5,000 for a Bronze contact.
- Set `NavigatorPosition = ednTop` to move the navigator above the fields. Or clear `Fields`, right click the data form and choose **Generate fields** to start from one item per dataset field.

5. Main components

The SGC Editors palette page is long, but a typical first form uses only a handful of its controls. This chapter lists them all in one table, then introduces them in groups with the properties you reach for first. The validator and its summary come from sgcUICore and are included because almost every form needs them.

Every class has its own help topic, with the properties, methods and events, their types and their defaults. Chapter 7 shows where to find them.

Reference table

Class	Palette page	What it is for	Key members
`TsgcVCLMonthCalendar`	SGC Editors	A month calendar, single date or date range	`Date`, `SelectionMode`, `RangeStart`, `RangeEnd`, `ShowWeekNumbers`, `OnSelect`
`TsgcVCLDatePicker`	SGC Editors	A date with a drop down calendar	`Date`, `MinDate`, `MaxDate`, `DisabledWeekDays`, `DisabledDates`
`TsgcVCLTimePicker`	SGC Editors	A time with a drop down list	`Time`, `MinTime`, `MaxTime`, `ListInterval`
`TsgcVCLDateTimePicker`	SGC Editors	A date and a time in one field	`DateTime`, `MinDate`, `MaxDate`, `MinTime`, `MaxTime`
`TsgcVCLSpinEdit`	SGC Editors	A number with arrows and a unit	`Value`, `MinValue`, `MaxValue`, `Increment`, `ValueType`, `Units`, `Wrap`
`TsgcVCLRangeSlider`	SGC Editors	A slider with one thumb, or two in range mode	`Value`, `EndValue`, `RangeMode`, `Min`, `Max`, `Step`, `LabelFormat`, `OnChangeComplete`
`TsgcVCLRating`	SGC Editors	Stars	`Value`, `Count`, `Precision`
`TsgcVCLCheckGroup`	SGC Editors	Several check boxes in one control	`Items` (`Caption`, `Value`, `Checked`), `Caption`
`TsgcVCLRadioGroup`	SGC Editors	One choice out of several	`Items`, `ItemIndex`, `Columns`, `Caption`
`TsgcVCLColorPicker`	SGC Editors	Swatches, hex, and in Registered builds an HSV square and alpha	`SelectedColor`, `Parts`, `Swatches`, `OnSelect`
`TsgcVCLTokenEdit`	SGC Editors	A list of entries shown as chips	`Tokens`, `Delimiters`, `MaxTokens`, `AllowDuplicates`, `OnValidateToken`
`TsgcVCLButtonEdit`	SGC Editors	A text edit with buttons	`Text`, `Buttons`, `OnButtonClick`
`TsgcVCLSearchEdit`	SGC Editors	A search box with a delay and a history	`Text`, `SearchDelay`, `History`, `OnSearch`

Class	Palette page	What it is for	Key members
`TsgcVCLMaskEdit`	SGC Editors	Text that follows a mask	`EditMask`, `Text`
`TsgcVCLCurrencyEdit`	SGC Editors	A money amount	`Value`, `Decimals`, `CurrencySymbol`, `AllowNegative`, `PositivePattern`
`TsgcVCLLookupEdit`	SGC Editors	A choice from a list with columns	`Items`, `Columns`, `KeyColumn`, `DisplayColumn`, `ColumnDelimiter`
`TsgcVCLMemo`	SGC Editors	Multi line text	`Lines`, `WordWrap`, `MaxLength`, `ShowCounter`, `AutoHeight`
`TsgcVCLNumPad`, `TsgcVCLNumPadPopup`	SGC Editors	A numeric keypad for touch screens	`Target`, `Associate`, `AutoPopup`, `OnCommit` (popup)
`TsgcVCLLayoutControl`	SGC Editors	Arranges controls in groups and tabs	`Items`, `SaveLayout`
`TsgcVCLDB...` (twelve classes)	SGC Editors	The data aware editors (Registered and Trial)	`DataSource`, `DataField`
`TsgcVCLDataForm`	SGC Editors	A generated data entry form (Registered and Trial)	`DataSource`, `Fields`, `NavigatorActions`, `OnEditorCreated`, `OnValidateField`
`TsgcVCLValidator`	SGC UICore	Checks the controls of a form	`Rules`, `Trigger`, `Display`, `Validate`, `ClearErrors`, `ErrorCount`, `OnValidated`
`TsgcVCLValidationSummary`	SGC UICore	Lists the errors of a validator	`Validator`, `Title`, `AutoHide`, `OnItemClick`

Dates and times

Four controls share the same date logic. `TsgcVCLMonthCalendar` is the calendar itself, always visible; set `SelectionMode` for a range and read `RangeStart` and `RangeEnd`. `TsgcVCLDatePicker` is an edit with the same calendar in a drop down, `TsgcVCLTimePicker` an edit with a list of times every `ListInterval` minutes, and `TsgcVCLDateTimePicker` both side by side.

- **Limits:** `MinDate`, `MaxDate`, `MinTime`, `MaxTime`, `DisabledWeekDays`, `DisabledDates`, and the `OnIsDateDisabled` event for your own rule, such as holidays from a database.
- **Look:** `ShowWeekNumbers`, `FirstDayOfWeek`, `DisplayFormat` on the pickers.
- **Typing:** the pickers accept typed dates and times. `OnParseError` lets you accept words of your own when a text cannot be read.

Numbers and ratings

`TsgcVCLSpinEdit` is the general number editor: whole numbers or fractions (`ValueType`, `Decimals`), a step (`Increment`), limits, a unit drawn after the number, and `Wrap` to jump from the maximum back to the minimum. `TsgcVCLRangeSlider` picks a value or, with `RangeMode`, a range by dragging; `ShowLabels` and

`LabelFormat` draw a scale. `TsgcVCLRating` shows `Count` stars and takes whole, half or exact values through `Precision`.

Choices, colours and tokens

- **`TsgcVCLCheckGroup`** and **`TsgcVCLRadioGroup`** hold their entries in an `Items` collection, each with a `Caption` for the user and a `Value` for your code. They follow the theme, which a standard `TCheckBox` or `TRadioGroup` does not.
- **`TsgcVCLColorPicker`** shows the `Parts` you choose: `ecpSwatches`, `ecpHex`, and in Registered and Trial builds `ecpHSV` and `ecpAlpha`. `SelectedColor` is an ARGB `TsgcUIColor`; `OnSelect` fires when the user picks one.
- **`TsgcVCLTokenEdit`** turns each entry into a chip, for tags, e-mail recipients or keywords. `Delimiters`, `MaxTokens`, `AllowDuplicates` and `OnValidateToken` set the rules, and Registered builds add an auto complete list.

Text editors

- **`TsgcVCLButtonEdit`**: a text edit with buttons inside it and `OnButtonClick`, for "choose a file" or "look up a customer" fields.
- **`TsgcVCLSearchEdit`**: a search box that fires `OnSearch` after `SearchDelay`, with a clear button and a history.
- **`TsgcVCLMaskEdit`**: text that follows an `EditMask`, with the same mask syntax as the standard `TMaskEdit`. An incomplete mask is an error the validator finds by itself.
- **`TsgcVCLCurrencyEdit`**: a `Currency` value with the symbol, the grouping and the decimals, formatted with the regional settings of Windows unless you set the patterns.
- **`TsgcVCLLookupEdit`**: a choice from `Items` with several columns, typed or picked from a drop down. A text that matches no entry is an error the validator finds by itself.
- **`TsgcVCLMemo`**: multi line text with an optional character counter and an automatic height between `MinLines` and `MaxLines`.
- **`TsgcVCLNumPad`** and **`TsgcVCLNumPadPopup`**: a keypad on the form or a popup keypad attached to an editor, for touch panels without a keyboard.

The validator and the summary

`TsgcVCLValidator` and `TsgcVCLValidationSummary` are described in chapter 3 and shown in demo 04. Remember three things: the rules live in `Rules`, the field names come from the labels with `FocusControl`, and `Validate` returns `True` when the form is valid. Both components are part of sgcUICore and of every edition.

Layout control, data form and data aware editors

- **`TsgcVCLLayoutControl`** arranges controls with captions in groups and tabs, and `SaveLayout` keeps the arrangement as JSON. It is part of every edition.
- **`TsgcVCLDataForm`** builds a layout control full of data aware editors from a `DataSource` and its `Fields`, with validation rules, a summary and a navigator. Demo 05 shows it. It can also edit an object of your own class instead of a dataset; the `DataForm` demo shows both.

- **The data aware editors**, `TsgcVCLDBDatePicker` to `TsgcVCLDBLookupEdit`, are the editors with `DataSource` and `DataField`, for forms you lay out by hand next to a `TDBGrid` and a `TDBNavigator`. The `DataAware` demo shows them.

The data form and the data aware editors are part of the Registered and Trial editions.

6. Common problems

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

Values and input

A value set in code is changed

What you see. You write `spnSeats.Value := 25` and the editor shows 20, or a slider ends a little below its `Max`.

Why. The value is clamped to the limits. If you set `Value` before `MinValue` and `MaxValue`, it is clamped to the old limits. The `Max` of a slider with a `Step` moves down to the step grid.

Fix. Set `MinValue`, `MaxValue`, `ValueType` and `Decimals` before `Value`. Remember that `MinValue` equal to `MaxValue` means no limit.

A typed date goes back to the previous value

What you see. The user types a date, leaves the field, and the old date comes back.

Why. The text could not be read, or the date is outside `MinDate` and `MaxDate`, or it is disabled by `DisabledDates`, `DisabledWeekDays` or `OnIsDateDisabled`. Demo 02 shows this: a date before today is refused because of `MinDate`.

Fix. Handle `OnParseError` to accept your own words, and call `IsDateEnabled` before you set a date in code.

OnChange runs when I did not expect it

What you see. Code in `OnChange` runs while the form loads its data, or a handler that writes another editor runs in a loop.

Why. `OnChange` fires for every change of the value, also for values written in code.

Fix. Use this on purpose, as demo 02 does to compute its total. When you load a record and do not want the handlers to run, set a flag of your own and check it at the top of the handler.

The amounts or dates look different from the screenshots

What you see. Your screen shows `$1,249.99` where this guide shows `1.249,99 $`, or another date order.

Why. The editors use the regional settings of Windows for separators, the position of the currency symbol and the date format. An empty `CurrencySymbol` also takes the symbol of Windows.

Fix. Nothing to fix: your code reads the same `Currency` and `TDate` values. If a fixed layout is required, set `CurrencySymbol`, `PositivePattern` and `NegativePattern` on the currency edit, or assign the `FormatSettings` of a control.

The time picker has no drop down button

Why. `ListInterval` is 0. **Fix.** Set it to the minutes between the items of the list.

Validation

The error messages show a strange field name

What you see. A message names the field by something other than its label, or the summary lines are hard to understand.

Why. The validator takes the field name from the `TLabel` whose `FocusControl` is the field. Without such a label it has no caption to use.

Fix. Set `FocusControl` on the label of every field, as all the QuickStart forms do, or give the rule an `ErrorMessage` of its own.

A field is invalid although no rule says so

What you see. A masked edit or a lookup edit gets a red border and a message, although `Rules` has no item for it, or the only rule is `uvrRequired` and the field is filled.

Why. With `AutoDiscover = True`, the default, the validator also runs the own checks of the sgcEditors controls: an incomplete mask, or a text that is not in the lookup list.

Fix. Complete the value, or set `AutoDiscover = False` if you want only your rules.

Nothing is checked while the user types

Why. `Trigger` is `uvtExit` by default, so a field is checked when the user leaves it. **Fix.** Set `Trigger = uvtChange`, or `uvtManual` to check only when you call `Validate`.

The data form

The data form is empty

What you see. At run time the data form shows no fields.

Why. The data form builds its editors when its dataset is active. A missing `DataSource`, a dataset without a `DataSet` or a closed dataset leaves it empty.

Fix. Check `DataSource` and open the dataset, as demo 05 does with `CreateDataSet`.

A generated editor needs a different size or setting

What you see. One of the generated editors is taller or shorter than you would like, or needs a property the field description does not have.

Why. The data form creates its editors at run time with its own defaults. For a choice of up to five options it creates a radio group with one column per option.

Fix. Handle `OnEditorCreated`. It receives the field description and the new control, so you can change the control before it is shown.

A data aware editor cannot be edited

Why. `ReadOnly` is `True`, the `DataSource` or its dataset is missing, or the dataset or the field is read only or calculated. **Fix.** Check each of these in turn.

Editions and compiling

Class TsgcVCLDBDatePicker not found, or TsgcVCLDataForm not found

What you see. A form loads with an error that a data aware class or the data form class is not found, or the compiler does not know `TsgcVCLDataForm`.

Why. The application is built with the Community edition, which has no data aware editors and no data form.

Fix. Build with the Registered or Trial edition. Demos 01 to 04 do not use these classes and build with every edition.

The colour picker has no square, or the slider shows one thumb

Why. `Parts` does not include `ecpHSV`, `RangeMode` is `False`, or the build is Community, which leaves out the HSV square, the alpha bar and the second thumb. The auto complete list of the token edit is left out in the same way.

Demo 05 asks for midas.dll

Why. A project with a `TClientDataSet` needs midas, either as `midas.dll` or linked into the executable. **Fix.** Keep `MidasLib` in the `uses` clause of the project, as demo 05 does.

7. Where next

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

The other Delphi demos

The other folders in `Demos\Editors\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 editor for the VCL on one form, with a theme switch, a read only switch and a log of every event they fire. The natural next step after demo 03.
`Calendar`	A month calendar with every option on the right: single date or date range, week numbers, disabled weekends and holidays, a minimum and a maximum date, and a locale switch.
`BookingForm`	A hotel booking form built from the editors: a date range on a calendar, time pickers with limits, a spin edit, a token edit with auto complete, a rating, a colour picker, a masked edit and a currency edit.
`Validation`	The validation framework of sgcUICore on one VCL form: every rule kind on standard and sgcEditors controls, the border, icon and summary display, on demand, on exit or on change. The natural next step after demo 04.
`DataAware`	The data aware editors on an in-memory `TClientDataSet`, next to a `TDBGrid` and a `TDBNavigator`, with a lookup on a second dataset and a validator.
`DataForm`	A layout control with groups, tabs and a splitter saved as JSON, and a data form generated from a `TClientDataSet` and from an object, with validation and Smart Paste. The natural next step after demo 05.
`GalleryFMX`	The editors for FireMonkey in one scrolling column, for Windows and Android, with a theme switch, a read only switch and an event log.
`ValidationFMX`	A sign-up form checked by `TsgcFMXValidator`, with sgcEditors controls and standard FireMonkey controls together, for Windows and Android.
`DataFormFMX`	The FireMonkey twin of the DataForm demo.

Some of these demos use features that are not part of every edition, such as the data aware editors, the data form, the second thumb of the slider or the auto complete list. Their readme files say what they need.

Help topics

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

Topic	What it covers
Introduction	What sgcEditors does and where to start.
Controls, Architecture	The catalogue of controls, and the models, hosts, inner edits and drop downs behind them.
Installation, Platforms	The packages, the Delphi versions and the target platforms.
Validation, Masks	Rules, triggers, display and summary; the mask syntax.
Themes, Localisation, Keyboard	Themes and accent colours, the `FormatSettings` of each control, the keys of every control and accessibility.
Layout, Data forms, Smart Paste	The layout control and the data form in depth.
Editions	Exactly what the Community edition leaves out.
Troubleshooting	More problems and their causes.
One topic per control	Under Components, with every property, its type and its default.

On the website

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

Enjoy building forms your users like to fill in.

---

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