sgcWizard

Getting Started

Guide your users through a task one page at a time, with steps, validation and a summary on a Delphi form

About this guide

This guide is for Delphi developers who know how to build a VCL form and want to add a step by step wizard to it with sgcWizard. You do not need any experience with wizard components. If you have ever clicked through an installer or a sign up form with Back and Next buttons, you already know what you are going to build.

The guide follows the five QuickStart demos. Each demo adds one idea to the previous one, from a wizard with three pages and no navigation code to a small onboarding application with validation and a generated summary page. Chapter 4 walks through them one at a time: what you will build, the components on the form, the code, how it works and a few changes to try.

Read it with Delphi open next to it. Open a demo, run it, click through the pages, then read its section and try the suggested changes. Chapters 1 to 3 explain what sgcWizard 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\Wizard\Delphi\00.QuickStart` folder of your sgcUI installation. Each demo has its own subfolder, from `01.HelloWizard` to `05.WizardSetupApp`, 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 Windows machine with Spanish regional settings. The demos show little more than text, but wherever a number or a date appears, it follows the Windows regional settings of the machine that runs the demo.

Contents

1. What sgcWizard is

The idea in one minute
Your first wizard needs no navigation code
What you need
Part of sgcUI
Coming from TPageControl and hand made wizards

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

A wizard is a page container
Pages are steps
The wizard paints the stepper and the buttons
Navigation: with events and without
Validation with OnBeforePageChange
Skipped pages
Finish and Cancel
The summary page
Themes, accent and transitions
Building a wizard in code

4. The QuickStart demos

4.1 Demo 01: Hello Wizard
4.2 Demo 02: Wizard Steps
4.3 Demo 03: Wizard Validation
4.4 Demo 04: Wizard Themes
4.5 Demo 05: Workspace Setup

5. Main components

Reference table
The wizard
The pages
The stepper on its own
Beyond the basics

6. Common problems

Designing the wizard
Navigation and events
Colours and themes
Running and testing

7. Where next

The other Delphi demos
Help topics
On the website
Getting help

1. What sgcWizard is

sgcWizard adds step by step wizards and step indicators to Delphi applications, for the VCL and for FireMonkey. A wizard splits a task into pages: an installer that asks for the licence, the folder and the components; a sign up form that asks for the account, the workspace and the plan; an order form that asks for the name, the delivery and the address. The user sees one page at a time, a band of numbered steps shows where they are, and Back, Next, Finish and Cancel buttons move between the pages.

You drop a `TsgcVCLWizard` on a form, add pages to it with a right click, and put ordinary Delphi controls on each page, exactly as you would on the tabs of a `TPageControl`. The wizard draws the step indicator and the button bar, handles the navigation and the keyboard, and fires events where your code checks the input, fills a page or saves the result.

This guide is for Delphi developers who are new to sgcWizard. By the end you will know how the pieces fit together, which properties and events 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 the last installer you clicked through. It had a few pages, each with one question. At the top or on the left it showed the list of steps, with the finished ones ticked and the current one highlighted. At the bottom it had a Back button, a Next button that turned into Finish (or Install) on the last page, and a Cancel button. Next refused to continue until you accepted the licence.

That is exactly what sgcWizard gives you, in three parts:

- **The pages.** Every page is a `TsgcVCLWizardPage`, a container like a `TPanel`, with your controls on it. Its `Caption` is the name of the step.
- **The stepper band.** One circle per page with the caption under it, drawn by the wizard. The active step is highlighted, the steps before it show a check mark, and a click on a completed step goes back to it.
- **The button bar.** Back, Next (Finish on the last page) and Cancel, also drawn by the wizard.

Your code only joins in where the wizard cannot know what to do: is the input on this page valid, what should the next page show, and what happens when the user clicks Finish or Cancel. That is the entire idea; everything else is detail.

Your first wizard needs no navigation code

The first QuickStart demo is the smallest wizard there is: three pages designed on the form, each with a title and a paragraph of text. In the form file the wizard and its pages look like this (the labels on the pages are left out):

```
object Wizard: TsgcVCLWizard
  Left = 0
  Top = 0
  Width = 640
  Height = 400
  Align = alClient
  TabOrder = 0
  OnFinish = WizardFinish
```

```
    OnCancel = WizardCancel
    object pgWelcome: TsgcVCLWizardPage
      Left = 0
      Top = 80
      Width = 640
      Height = 268
      Caption = 'Welcome'
      // ...
    end
    object pgDetails: TsgcVCLWizardPage
      // ...
      Caption = 'Details'
      // ...
    end
    object pgDone: TsgcVCLWizardPage
      // ...
      Caption = 'Done'
      // ...
    end
  end
```

And this is all the code of the demo. Next, Back and the clicks on the steps need nothing; only the two buttons that end the wizard are answered:

```
procedure TfrmHelloWizard.WizardFinish(Sender: TObject);
begin
  { OnFinish fires when Finish (the Next button of the last page) is
    clicked. A real wizard saves its data here. }
  ShowMessage('The wizard is finished.');
  Close;
end;

procedure TfrmHelloWizard.WizardCancel(Sender: TObject);
begin
  { OnCancel fires when Cancel is clicked or Escape is pressed. }
  Close;
end;
```

Run it and you get a three step wizard with a highlighted current step, Back disabled on the first page, Next turning into Finish on the last one, Enter for Next and Escape for Cancel. Section 4.1 walks through the demo in detail.

What you need

- **Delphi 7 or later.** The packages cover Delphi 7, 2007, 2009, 2010, XE to XE8, 10 Seattle to 10.4 Sydney, 11, 12 and 13.
- **VCL on Windows.** `TsgcVCLWizard`, `TsgcVCLWizardPage` and `TsgcVCLStepper` work in every Delphi version, for Win32, and for Win64 from XE2 on.
- **FireMonkey, from Delphi XE6.** `TsgcFMXWizard`, `TsgcFMXWizardPage` and `TsgcFMXStepper` have the same API and run on Windows, macOS, iOS and Android, from XE6 to 10.2 on Windows only.
- **sgcUICore**, the shared runtime of the sgcUI products. It is installed together with sgcWizard.

This guide and its demos use the VCL. Once you know the VCL wizard, the FireMonkey one holds no surprises: chapter 7 points you to the FireMonkey version of the onboarding demo.

Part of sgcUI

sgcWizard is one product of sgcUI, the eSeGeCe suite of visual components, next to products such as sgcGrid, sgcDashboard, sgcNavigation and sgcToolbar. All of them are drawn with sgcUICore, which provides the canvas, the fonts and the light and dark themes. That has two practical consequences:

- A wizard looks the same on the VCL and on FireMonkey, and it matches the other sgcUI controls on the same form.
- One call such as `sgcUISetApplicationTheme(utDark, True)` switches every sgcUI control of the application to the dark theme at once, the wizard included. Demo 04 shows it.

You do not need any other sgcUI product to use sgcWizard.

Coming from TPageControl and hand made wizards

Many Delphi wizards are built by hand: a `TPageControl` with hidden tabs, a panel with three buttons at the bottom, and code that changes `ActivePageIndex`, enables and disables the buttons, renames Next to Finish on the last tab, and draws a list of steps with labels or images. It works, but every wizard repeats the same plumbing.

sgcWizard keeps the part you know, a container with pages that you design in the form designer, and takes over the plumbing:

By hand	With sgcWizard
A `TPageControl` with its tabs hidden	A `TsgcVCLWizard` with `TsgcVCLWizardPage` pages
Labels or images for the list of steps	The stepper band, drawn from the page captions
Three `TButton` controls and their enabled state	The button bar, with Next turning into Finish on the last page
Checks in the OnClick of the Next button	`OnBeforePageChange` with `aAllow`, fired for every kind of move
Code that jumps over a page that does not apply	`Skip := True` on that page
A review page built by hand	`ShowSummary := True`

The controls on the pages stay ordinary VCL controls, so moving an existing hand made wizard over is mostly a matter of moving the controls from the tabs to the pages.

2. Install and the palette

This chapter gets sgcWizard 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

sgcWizard 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 **sgcWizard**. The installer shows one checkbox per product and enables the products your licence includes. sgcUICore has no checkbox: it is installed automatically with sgcWizard.
4. Leave **Demo Samples** selected, so the QuickStart demos are copied too.
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 sgcWizard units without any setting on your side.

The packages

You only need this section if you build the packages yourself. sgcWizard 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
`sgcWizard<V>`	Runtime	The stepper model, the renderer with the PNG and SVG output, the strings and the accessibility support. No VCL, no FireMonkey. Requires sgcUICore.
`sgcWizardVCL<V>`	Runtime	`TsgcVCLWizard`, `TsgcVCLWizardPage` and `TsgcVCLStepper`.
`sgcWizardFMX<V>`	Runtime, XE6 and later	`TsgcFMXWizard`, `TsgcFMXWizardPage` and `TsgcFMXStepper`.
`dclsgcWizard<V>`	Design time	Registers the VCL wizard and stepper on the palette, with the component editor that adds pages.
`dclsgcWizardFMX<V>`	Design time, XE6 and later	Registers the FireMonkey controls. Install `dclsgcWizard` first.

Build them in dependency order: sgcUICore first, then `sgcWizard`, then `sgcWizardVCL`, then `dclsgcWizard`. 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 Wizard**. In a VCL project it shows two components:

Component	Visual	What it does
`TsgcVCLWizard`	yes	The wizard: a page container with a stepper band and a button bar.
`TsgcVCLStepper`	yes	The step indicator on its own, horizontal or vertical, for any flow that needs to show its progress.

In a FireMonkey project the same page shows `TsgcFMXWizard` and `TsgcFMXStepper` instead. The QuickStart demos use only `TsgcVCLWizard`.

> **Note.** You will not find `TsgcVCLWizardPage` on the palette. A page can only live on a wizard, so it is added with the component editor of the wizard, described next.

Help at design time

Right click a wizard, or one of its pages, in the form designer to find three commands:

- **Add Page** adds a new page at the end of the wizard and selects it. This is how every page of the demos was created.
- **Next Page** and **Previous Page** show the next or the previous page in the designer, so you can put controls on it.

A click on a step of the stepper band in the designer also brings its page to the front. Select a page by clicking an empty area of it, and its `Caption`, `Description`, `StepGlyph` and `Skip` appear in the Object Inspector.

Editions

sgcWizard is built from the same source for every edition, and every edition has every feature: the VCL and FireMonkey wizards, pages and steppers, validation, skipped pages, the summary page, the themes, the transitions, the design time support and the PNG and SVG output. The trial edition is meant for evaluation. Everything this guide shows works the same in every edition.

Where the demos are

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

Folder	Project	What it shows
01.HelloWizard	`HelloWizard.dpr`	The smallest wizard: three pages designed on the form, no navigation code
02.WizardSteps	`WizardSteps.dpr`	The look of the steps and the buttons, set in the Object Inspector: an installer layout
03.WizardValidation	`WizardValidation.dpr`	Validation that refuses Next, a page that is skipped, a page filled before it is shown
04.WizardThemes	`WizardThemes.dpr`	Light and dark themes, the Classic and Fluent styles, the accent colour and the page transitions
05.WizardSetupApp	`WizardSetupApp.dpr`	A small onboarding application with validation and a generated summary page

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

Running your first demo

1. In Delphi, choose **File** > **Open Project** and open `Demos\Wizard\Delphi\00.QuickStart\01.HelloWizard\HelloWizard.dpr`.
2. Press **F9**. The window opens on the Welcome page.
3. Click **Next** twice, then **Back**, then the first step at the top. Press **Enter** and **Escape** and see what happens.
4. Back in the IDE, open the form, right click the wizard and choose **Next Page** to look at the other pages in the designer.

> **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

A handful of ideas explain almost everything a wizard does. Read this chapter once before the demos, and come back to it when something behaves differently from what you expected.

A wizard is a page container

`TsgcVCLWizard` works like a `TPageControl`. Its pages are child controls, kept in the order they were added, and only the active page is visible. A page is a `TsgcVCLWizardPage`: a plain container on which you drop labels, edits, check boxes, combo boxes, memos, panels, frames or any other control.

Three members tell you where the user is:

Member	What it is
`ActivePageIndex`, `ActivePage`	The index of the visible page, starting at 0, and the page itself. `ActivePageIndex` is published, so the page shown when the form opens can be set in the Object Inspector.
`PageCount`, `Pages[i]`	The number of pages and the page at an index.
`PageIndex` (of a page)	The position of a page in its wizard. Comparing with `pgAddress.PageIndex` is more robust than comparing with a fixed number, because the numbers change when you reorder the pages.

A page can only be placed on a wizard. Setting its `Parent` to a panel or a form raises `EsgcWizardError`. To group controls on a page, put a `TPanel` or a frame on the page.

Pages are steps

Every page is one step of the stepper band, and its properties describe the step:

Page property	What it does
`Caption`	The name of the step, under or next to its circle.
`Description`	A second, smaller line under the caption, for example “Where to install”.
`StepGlyph`	A short text drawn in the circle instead of the step number, such as `i` or `OK`. A completed step shows a check mark either way.
`Skip`	The page stays in the wizard, but Next and Back jump over it and its step is drawn as skipped.

The states of the steps follow the wizard by themselves: the active step is highlighted, the steps before it are completed, the steps after it are pending. You never set a step state on a wizard.

The wizard paints the stepper and the buttons

The wizard owns two internal controls that you cannot select in the designer and that are not stored in the form file:

- **The stepper band.** `StepperPosition` puts it at the top (`wspTop`, the default), on the left as a vertical list (`wspLeft`), or hides it (`wspNone`).
- **The button bar.** Back, Next, and Cancel while `ShowCancel` is True. On the last page Next becomes Finish. The texts come from `BackCaption`, `NextCaption`, `FinishCaption` and `CancelCaption`. Left empty, they are localized defaults that follow the language of the application.

Both are available at run time as the `Stepper` and `ButtonBar` properties, but a beginner never needs them. The wizard also answers the keyboard: **Enter** is Next (or Finish), **Escape** is Cancel, and **Ctrl+Tab** and **Ctrl+Shift+Tab** move to the next and the previous step.

Navigation: with events and without

There are two ways to change the page from code, and the difference matters:

Call	Events	Use it for
`Next`, `Back`, `GoToPage(i)`, `Finish`	Yes: `OnBeforePageChange`, then `OnPageChanged` (or `OnFinish`)	Moving as if the user had clicked. Skipped pages are jumped over and the validation runs. Each returns True when the page changed.
`ActivePageIndex := i`, `ActivePage := pg`	No	Choosing the page shown when the form opens, or resetting the wizard.

The events in the order they fire for a move forward:

1. `OnBeforePageChange(Sender, aFromIndex, aToIndex, var aAllow)`: set `aAllow` to False to stay on the page. It fires for Next, Back, a click on a step, Ctrl+Tab and Finish.
2. The page changes, with an animation if `Transition` is set (`OnTransitionStart` and `OnTransitionEnd` bracket it).
3. `OnPageChanged(Sender)`: the new page is `ActivePage`. This is the place to fill a page from the answers of the previous ones.

Validation with OnBeforePageChange

`OnBeforePageChange` is the single place where you check the input. A typical handler lets every move backwards through, checks the page being left on a move forward, and shows a message when the input is not valid. This one is from demo 03:

```
{ only a move forward is checked, Back is always allowed }
if aToIndex < aFromIndex then
  Exit;
// ...
if aFromIndex = pgName.PageIndex then
begin
  aAllow := Trim(edtName.Text) <> '';
  if aAllow then
    lblNameError.Caption := ''
  else
    lblNameError.Caption := 'Enter your name to continue.';
end
```

On Finish the event fires too, with `aToIndex` equal to `PageCount`, one past the last page. That lets you refuse Finish in the same handler, for example while a licence is not accepted:

```
  { Finish arrives here too, with aToIndex = PageCount }
  if aToIndex = Wizard.PageCount then
    aAllow := chkAccept.Checked;
```

Skipped pages

Some pages only apply sometimes: the delivery address when the customer picks the order up in the store, the proxy settings when there is no proxy. Do not remove such a page; set its `Skip` property. Next and Back jump over a skipped page, its step is drawn as skipped, and as soon as `Skip` is False again the page is back in the flow. Demo 03 sets it from two radio buttons:

```
procedure TfrmWizardValidation.rbDeliveryClick(Sender: TObject);
begin
  { a skipped page stays in the wizard, Next and Back jump over it }
  pgAddress.Skip := rbStore.Checked;
end;
```

When every page after the active one is skipped, the active page is the last one reachable, and its Next button becomes Finish.

Finish and Cancel

The wizard never closes anything by itself. Finish fires `OnFinish` and Cancel (or Escape) fires `OnCancel`; what happens next is up to you. In the demos both handlers call `Close`. When the wizard sits on a form shown with `ShowModal`, set `ModalResult` instead, so the caller knows how the wizard ended:

```
procedure TfrmSnippets.WizardFinish(Sender: TObject);
begin
  ModalResult := mrOk;
end;

procedure TfrmSnippets.WizardCancel(Sender: TObject);
begin
  ModalResult := mrCancel;
end;
```

The summary page

Many wizards end with a review page: here is what you entered, click Finish to go ahead. Set `ShowSummary` to True and the wizard generates that page for you, after the last page you designed. You do not design it, and it is not stored in the form file. It lists the answers of every visited page, one section per page with an **Edit** link that goes back to that page; Next on that page then returns straight to the summary.

The summary finds the answers on its own. For every input control on a page it needs a caption and a value:

- **The caption** comes from a `TLabel` whose `FocusControl` points at the control, without the trailing colon (“Name:” becomes “Name”), or from the control's own caption, as with a check box.

- **The value** is the text of an edit, the selected item of a combo box, or Yes or No for a check box.

So setting `FocusControl` on the labels of your pages is all the summary needs. `SummaryTitle` sets the heading of the page and `SummaryCaption` the name of its step. To add, change or delete lines, handle `OnGetSummaryItems`: it receives the page and the lines found on it, as a `TsgcWizardSummaryItems` with `Add`, `IndexOf` and `Delete`. Demo 05 adds a computed price this way.

Themes, accent and transitions

The wizard is drawn with the sgcUICore themes. Its `Theme` property decides which one:

- `utDefault` (the default) follows the application theme. Set the application theme once, with `sgcUISetApplicationTheme(utDark, True)` from the unit `sgcUICore_Theme`, and every sgcUI control follows it. The second parameter, True, also themes the standard VCL controls of every form. `sgcUISetApplicationThemePreset(CS_UI_THEME_FLUENT)` switches the style from Classic to Fluent, the look of Windows 11.
- `utLight` or `utDark` pins this one wizard to that theme, whatever the application does.

`AccentColor` colours the active step, the connectors of the completed steps and the Next button. It is an ARGB value, `$AARRGGBB` with the alpha first, so a solid purple is `$FF7C3AED`; 0 uses the accent of the theme.

The wizard also themes the standard controls on its pages: labels, edits, check boxes and the rest get the text and background colours of the theme when the form loads and on every theme change. That is what keeps a dark wizard readable, and it is also why a label coloured red in the Object Inspector comes out in the theme text colour. Chapter 6 shows the fix.

Finally, `Transition` animates the page changes: `wtSlide` slides the new page in (from the right on Next, from the left on Back), `wtFade` fades it in, `wtSlideFade` does both, and `wtNone`, the default, switches at once. `TransitionDuration` is the length in milliseconds. `ReduceMotion` left at `wrmSystem` turns the animation off when Windows is set to reduce animations; `wrmReduce` always turns it off and `wrmAnimate` always animates.

Building a wizard in code

The demos design their pages in the form designer, which is the easiest way. When the pages depend on data known only at run time, create them in code with `AddPage`. It creates a page, sets its caption and puts it on the wizard; then you create the controls on the page as usual:

```
procedure TfrmSnippets.BuildWizard;
var
  vEdit: TEdit;
begin
  Wizard := TsgcVCLWizard.Create(Self);
  Wizard.Parent := Self;
  Wizard.Align := alClient;
  Wizard.OnFinish := WizardFinish;
  Wizard.OnCancel := WizardCancel;

  { AddPage creates the page, sets its Caption and puts it on the wizard }
  pgAccount := Wizard.AddPage('Account');
  pgAccount.Description := 'Who you are';
  vEdit := TEdit.Create(Self);
  vEdit.Parent := pgAccount;
  vEdit.SetBounds(32, 32, 240, 23);

  Wizard.AddPage('Done');
  // ...
end;
```

Here `Wizard` and `pgAccount` are fields of the form, and the units in the `uses` clause are `Classes`, `Controls`, `Forms`, `StdCtrls` and `sgcWizard_VCL`. Controls added to a page in code are themed like the ones designed on the form.

4. The QuickStart demos

The five demos build on each other. Run them in order: each one adds a few ideas to the previous one and explains them in the comments of its unit. This chapter repeats the essentials, with the components, the code and some changes to try.

Demo	Project	New ideas
01.HelloWizard	`HelloWizard.dpr`	The wizard as a page container, the Add Page command, OnFinish and OnCancel
02.WizardSteps	`WizardSteps.dpr`	Caption, Description and StepGlyph of a page, the stepper position, the button captions, ActivePageIndex, OnPageChanged
03.WizardValidation	`WizardValidation.dpr`	OnBeforePageChange and aAllow, Skip, filling a page in OnPageChanged, Next called from code
04.WizardThemes	`WizardThemes.dpr`	The application theme and style, AccentColor, Transition
05.WizardSetupApp	`WizardSetupApp.dpr`	ShowSummary and FocusControl, OnGetSummaryItems, validation per page, a complete flow

> **Note.** The demos need sgcWizard installed, so the components appear on the SGC Wizard palette page and the `sgcUI\Source` folder is on the library path. They compile with every Delphi version from Delphi 7 on, for Win32 and Win64.

4.1 Demo 01: Hello Wizard

Folder `01.HelloWizard`, project `HelloWizard.dpr`.

What you will build

The smallest wizard there is. Three pages, Welcome, Details and Done, each with a bold title and a paragraph of text, all designed on the form. The wizard draws the steps at the top and the buttons at the bottom, and moves between the pages without a single line of navigation code.

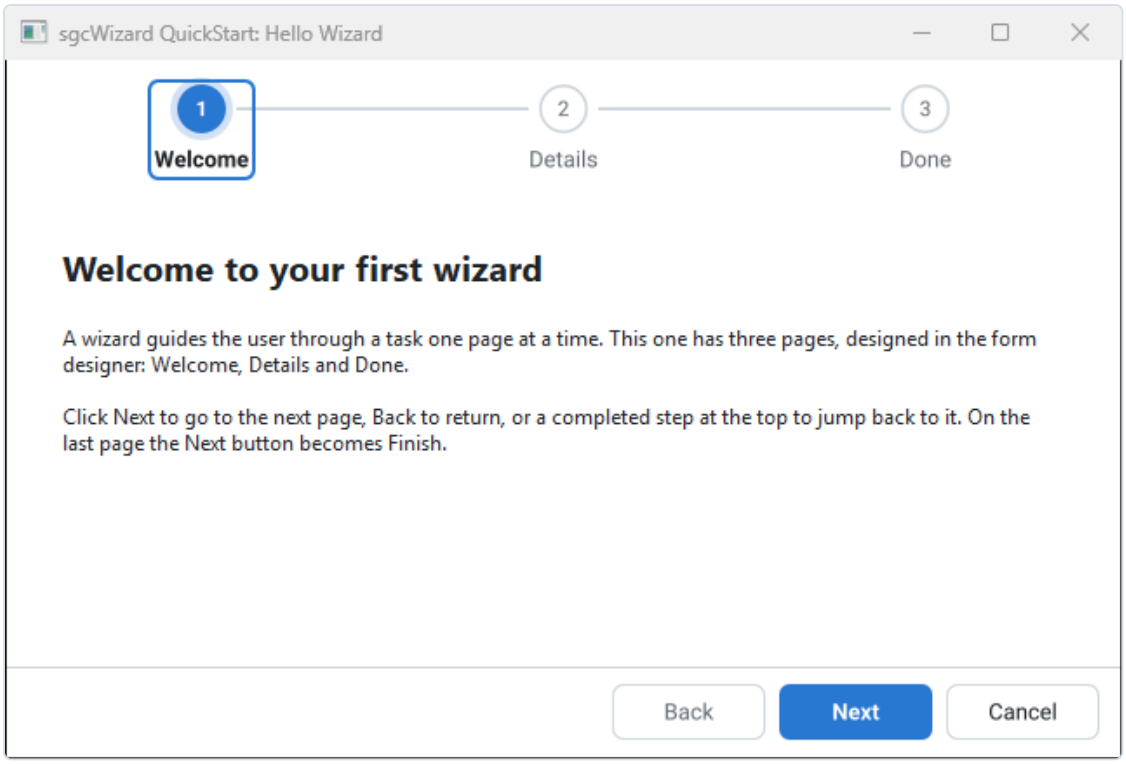


Figure 4.1. Demo 01 on its first page. Back is disabled because there is no page before Welcome.

Components on the form

Component	Class	Properties you set
`Wizard`	`TsgcVCLWizard`	`Align = alClient`, `OnFinish`, `OnCancel`. Nothing else: every other property keeps its default.
`pgWelcome`, `pgDetails`, `pgDone`	`TsgcVCLWizardPage`	`Caption` = Welcome, Details, Done. Added with **Add Page**.
`lblWelcomeTitle` and the other titles	`TLabel`	Bold Segoe UI, 20 pixels.
`lblWelcomeText` and the other texts	`TLabel`	`AutoSize = False`, `WordWrap = True`.

A page in the form file is an ordinary container with its controls inside it:

```
object pgDone: TsgcVCLWizardPage
  Left = 0
  Top = 80
  Width = 640
  Height = 268
  Caption = 'Done'
  object lblDoneTitle: TLabel
    // ...
    Caption = 'All finished'
    // ...
  end
  object lblDoneText: TLabel
    // ...
    AutoSize = False
    Caption =
      'This is the last page, so the Next button now reads Finish. Cli' +
```

```
          'ck it to fire the OnFinish event of the wizard.'
        WordWrap = True
      end
    end
```

The code

The `uses` clause names one sgcWizard unit, `sgcWizard_VCL`, which the IDE adds by itself when you drop the wizard:

```
uses
  Classes, Controls, Forms, Dialogs, StdCtrls,
  // sgc
  sgcWizard_VCL;
```

And these are the only two event handlers of the demo:

```
procedure TfrmHelloWizard.WizardFinish(Sender: TObject);
begin
  { OnFinish fires when Finish (the Next button of the last page) is
    clicked. A real wizard saves its data here. }
  ShowMessage('The wizard is finished.');
  Close;
end;

procedure TfrmHelloWizard.WizardCancel(Sender: TObject);
begin
  { OnCancel fires when Cancel is clicked or Escape is pressed. }
  Close;
end;
```

How it works

1. The form loads from its DFM. The wizard reads its three pages in the order they are stored, builds one step per page from the page captions and shows the first page, because `ActivePageIndex` is 0.
2. Next calls `Wizard.Next`. The wizard hides the active page, shows the next one, marks the previous step as completed and highlights the new one. Back does the same in the other direction.
3. A click on a completed step at the top goes back to that page. Steps that were never reached cannot be clicked.
4. On the last page the Next button shows Finish. Clicking it, or pressing Enter, fires `OnFinish`; the demo shows a message and closes the form.
5. Cancel, or Escape, fires `OnCancel`, and the demo closes the form.

Try this

- Right click the wizard, choose **Add Page**, give the new page a `Caption` and a label. Run again: the stepper has four steps.
- Set `ShowCancel` to False. The Cancel button disappears. Escape still fires `OnCancel`, so keep the handler.
- Run the demo and drive it with the keyboard only: Enter, Escape, Ctrl+Tab and Ctrl+Shift+Tab.

4.2 Demo 02: Wizard Steps

Folder `02.WizardSteps`, project `WizardSteps.dpr`.

What you will build

The classic installer layout: a vertical list of steps on the left, a licence, a folder, the components to install and a Ready page. Every part of the look is set in the Object Inspector: a second line under each step, a text in the circle instead of the number, and the captions of the buttons. The window opens on the second page, and its title bar always says which step is active.

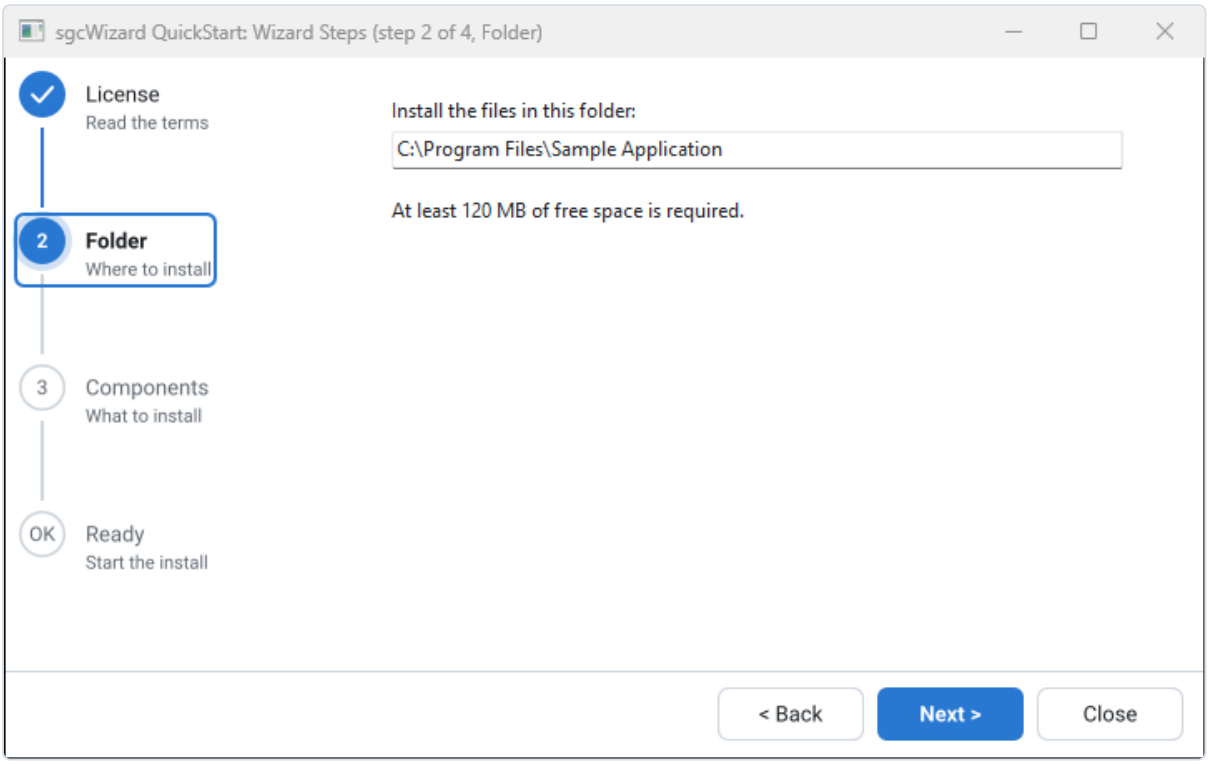


Figure 4.2. Demo 02. License shows a check mark because it is completed; Ready shows its `StepGlyph`, OK, instead of the number 4.

Components on the form

Component	Class	Properties you set
`Wizard`	`TsgcVCLWizard`	`ActivePageIndex = 1`, `StepperPosition = wspLeft`, `BackCaption = '< Back'`, `NextCaption = 'Next >'`, `FinishCaption = 'Install'`, `CancelCaption = 'Close'`, `OnPageChanged`, `OnFinish`, `OnCancel`.
`pgLicense`	`TsgcVCLWizardPage`	`Caption = 'License'`, `Description = 'Read the terms'`, `StepGlyph = 'i'`. A read only `TMemo` and the check box `chkAccept`.
`pgFolder`	`TsgcVCLWizardPage`	`Description = 'Where to install'`. The label `lblFolder` (`FocusControl = edtFolder`), the edit `edtFolder` and `lblSpace`.
`pgComponents`	`TsgcVCLWizardPage`	`Description = 'What to install'`. Four check boxes.
`pgReady`	`TsgcVCLWizardPage`	`Description = 'Start the install'`, `StepGlyph = 'OK'`. The label `lblReady`.

The wizard and its first page in the form file:

```
  object Wizard: TsgcVCLWizard
    // ...
    Align = alClient
    // ...
    ActivePageIndex = 1
    StepperPosition = wspLeft
    BackCaption = '< Back'
    NextCaption = 'Next >'
    FinishCaption = 'Install'
    CancelCaption = 'Close'
    OnPageChanged = WizardPageChanged
    OnFinish = WizardFinish
    OnCancel = WizardCancel
    object pgLicense: TsgcVCLWizardPage
      // ...
      Caption = 'License'
      Description = 'Read the terms'
      StepGlyph = 'i'
      // ...
```

The code

```
procedure TfrmWizardSteps.FormCreate(Sender: TObject);
begin
  { ActivePageIndex was set in the DFM, which switches the page without
    events, so the caption is set once here. }
  WizardPageChanged(Wizard);
end;

procedure TfrmWizardSteps.WizardPageChanged(Sender: TObject);
begin
  { OnPageChanged fires after Next, Back or a click on a step. }
  Caption := Format('sgcWizard QuickStart: Wizard Steps (step %d of %d, %s)',
    [Wizard.ActivePage.PageIndex + 1, Wizard.PageCount,
    Wizard.ActivePage.Caption]);
end;
```

How it works

1. With `StepperPosition = wspLeft` the stepper band is a vertical list on the left. Each step shows the page `Caption` in bold and the `Description` under it.
2. `ActivePageIndex = 1` opens the wizard on the Folder page. The License step before it is drawn as completed, with a check mark instead of its glyph.
3. Setting `ActivePageIndex` does not fire `OnPageChanged`, so `FormCreate` calls the handler once to put the step into the title bar.
4. From then on, every Next, Back or click on a step fires `OnPageChanged`. `ActivePage.PageIndex` is zero based, hence the `+ 1`.
5. On the last page the Next button shows the `FinishCaption`, Install.

Try this

- Set `StepperPosition` to `wspTop`, then to `wspNone`. With `wspNone` the wizard is only pages and buttons.
- Clear `NextCaption` in the Object Inspector. The button shows the localized default text again.
- Run the demo and click the completed License step to go back to it.

4.3 Demo 03: Wizard Validation

Folder `03.WizardValidation`, project `WizardValidation.dpr`.

What you will build

A small order form with four pages: the customer name, home delivery or pick up in the store, the delivery address, and a confirmation. Next is refused while the name or the street is empty, with a red message on the page. When the customer picks the order up in the store, the Address page is skipped. The Confirm page shows a summary of the order, built in code when the page is shown.

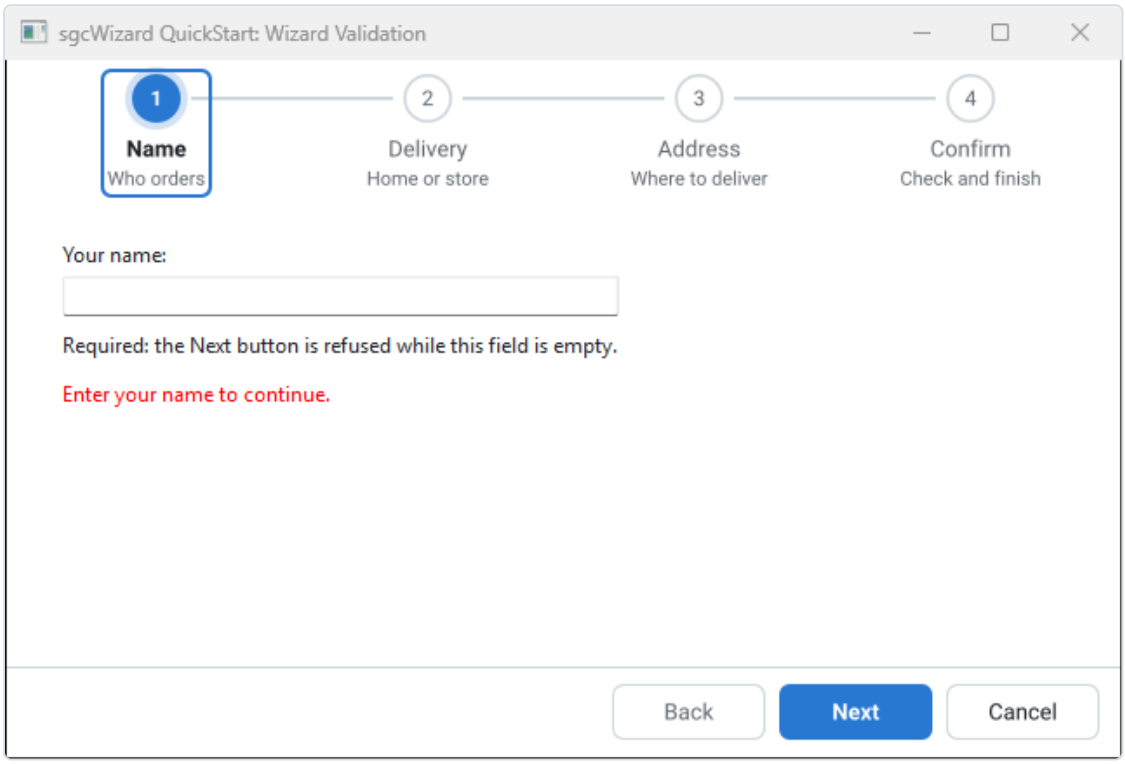


Figure 4.3. Demo 03 opens with the name empty and the refusal already shown. Each step has a `Description`.

Components on the form

Component	Class	Properties you set
`Wizard`	`TsgcVCLWizard`	`OnBeforePageChange`, `OnPageChanged`, `OnFinish`, `OnCancel`.
`pgName`	`TsgcVCLWizardPage`	`Caption = 'Name'`, `Description = 'Who orders'`. The edit `edtName`, a hint label and the error label `lblNameError`.
`pgDelivery`	`TsgcVCLWizardPage`	`Description = 'Home or store'`. The radio buttons `rbHome` (checked) and `rbStore`, both with `OnClick = rbDeliveryClick`.
`pgAddress`	`TsgcVCLWizardPage`	`Description = 'Where to deliver'`. The edits `edtStreet` and `edtCity` and the error label `lblAddressError`.
`pgConfirm`	`TsgcVCLWizardPage`	`Description = 'Check and finish'`. The label `lblConfirm`, filled in code.

The code

The form opens by calling `Next` once. The name is still empty, so the validation refuses the move and the window opens with the red message:

```
procedure TfrmWizardValidation.FormCreate(Sender: TObject);
begin
  { Next can also be called in code, it runs the same validation as the
    button. The demo calls it once with the name still empty, so the window
    opens with the refusal and its red message. Type a name and click Next. }
  Wizard.Next;
end;
```

The validation itself, in `OnBeforePageChange`:

```
procedure TfrmWizardValidation.WizardBeforePageChange(Sender: TObject;
  aFromIndex, aToIndex: Integer; var aAllow: Boolean);
begin
  { only a move forward is checked, Back is always allowed }
  if aToIndex < aFromIndex then
    Exit;

  { the wizard paints the controls of its pages with the text colour of its
    theme, so the red of the error labels is set here, in code }
  lblNameError.Font.Color := clRed;
  lblAddressError.Font.Color := clRed;

  { the page indexes follow the order of the pages in the wizard, PageIndex
    gives the index of a page }
  if aFromIndex = pgName.PageIndex then
  begin
    aAllow := Trim(edtName.Text) <> '';
    if aAllow then
      lblNameError.Caption := ''
    else
      lblNameError.Caption := 'Enter your name to continue.';
  end
  else if aFromIndex = pgAddress.PageIndex then
  begin
    aAllow := Trim(edtStreet.Text) <> '';
    if aAllow then
      lblAddressError.Caption := ''
    else
      lblAddressError.Caption := 'Enter the street for the delivery.';
  end;
end;
```

The skipped page and the confirmation:

```
procedure TfrmWizardValidation.rbDeliveryClick(Sender: TObject);
begin
  { a skipped page stays in the wizard, Next and Back jump over it }
  pgAddress.Skip := rbStore.Checked;
end;

function TfrmWizardValidation.DoOrderText: string;
begin
  Result := 'Name: ' + Trim(edtName.Text) + sLineBreak;
  if rbStore.Checked then
    Result := Result + 'Delivery: pick up in the store'
  else
    Result := Result + 'Delivery: home delivery' + sLineBreak + 'Address: ' +
      Trim(edtStreet.Text) + ', ' + Trim(edtCity.Text);
end;
```

```
procedure TfrmWizardValidation.WizardPageChanged(Sender: TObject);
begin
  { ActivePage is the page now shown }
  if Wizard.ActivePage = pgConfirm then
    lblConfirm.Caption := 'Check your order and click Finish.' + sLineBreak +
      sLineBreak + DoOrderText;
end;

procedure TfrmWizardValidation.WizardFinish(Sender: TObject);
begin
  ShowMessage('Order placed.' + sLineBreak + sLineBreak + DoOrderText);
  Close;
end;
```

How it works

1. Before every page change the wizard calls `OnBeforePageChange` with the index of the page being left, `aFromIndex`, and of the page it wants to show, `aToIndex`. `aAllow` starts as True.
2. A move backwards has `aToIndex < aFromIndex` and is let through at once, so the user can always go back and change an answer.
3. On a move forward the handler checks the page being left. Leaving the name empty sets `aAllow` to False: the page does not change and the red message appears.
4. Choosing “Pick up in the store” sets `pgAddress.Skip`. Next on the Delivery page then goes straight to Confirm, and the Address step is drawn as skipped. The address validation never runs, because the Address page is never left.
5. When the Confirm page is shown, `OnPageChanged` fills its label from the answers of the previous pages.

Try this

- Require the city too: add a check of `edtCity.Text` in the `pgAddress` branch.
- Refuse Finish in the same handler. Finish fires `OnBeforePageChange` with `aToIndex = Wizard.PageCount`.
- Add a button that calls `Wizard.GoToPage(pgConfirm.PageIndex)`. The validation runs for that jump as well.

4.4 Demo 04: Wizard Themes

Folder `04.WizardThemes`, project `WizardThemes.dpr`.

What you will build

A three page profile wizard with a row of combo boxes above it that change its look while it runs: light or dark theme, Classic or Fluent style, the accent colour, and the animation of the page changes. It opens in the dark theme with a purple accent and a slide transition.

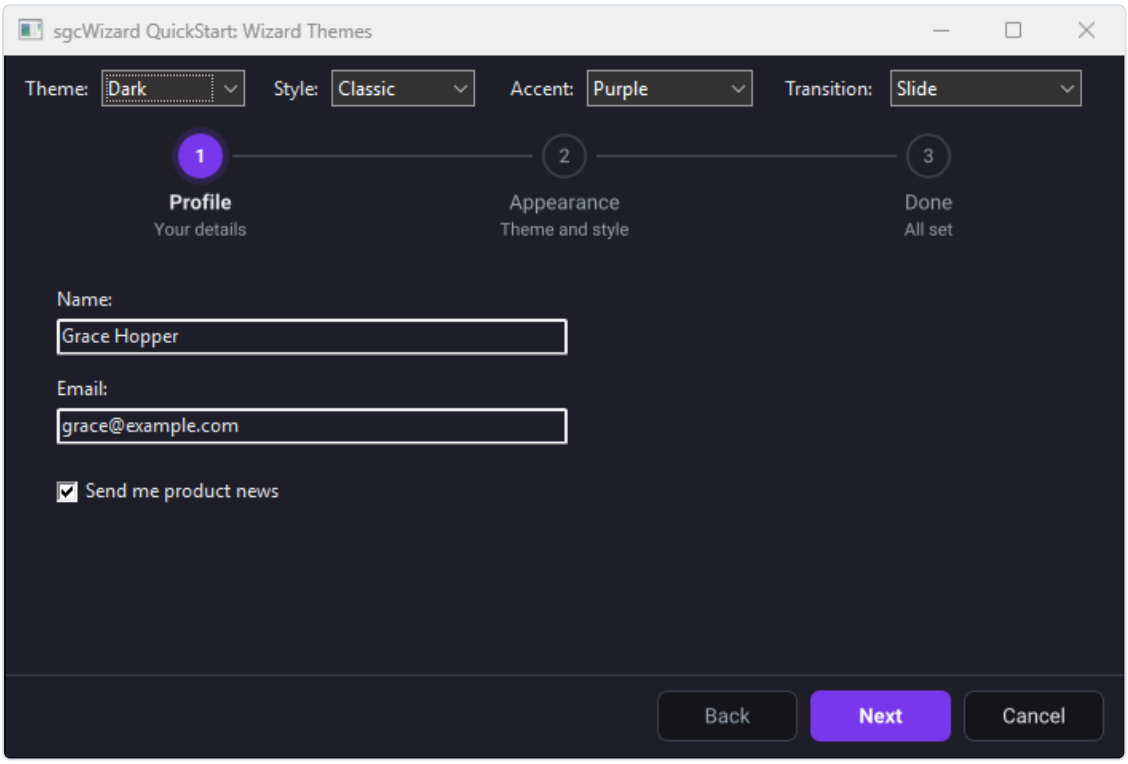


Figure 4.4. Demo 04 in the dark theme with the purple accent. The labels, edits and the top panel are themed as well.

Components on the form

Component	Class	Properties you set
`pnlTop`	`TPanel`	`Align = alTop`. Holds four labelled combo boxes: `cboTheme` (Light, Dark; `ItemIndex = 1`), `cboStyle` (Classic, Fluent), `cboAccent` (Theme, Purple, Green, Orange; `ItemIndex = 1`) and `cboTransition` (None, Slide, Fade, Slide + fade; `ItemIndex = 1`), each with an `OnChange` handler.
`Wizard`	`TsgcVCLWizard`	`Align = alClient`, `AccentColor = -8635667` (the ARGB value `$FF7C3AED`, purple), `Transition = wtSlide`, `Theme` left at `utDefault`, `OnFinish`, `OnCancel`.
`pgProfile`, `pgLook`, `pgDone`	`TsgcVCLWizardPage`	Captions Profile, Appearance and Done. The Profile page has two labelled edits and a check box.

The Object Inspector shows `AccentColor` as a plain integer, because a 32 bit ARGB value with the alpha byte set is a negative number. In code, write it in hexadecimal.

The code

The `uses` clause adds two units: `sgcUICore_Theme` for the application theme and `sgcWizard_Model` for the `wtSlide` family of values:

```
uses
  Classes, Controls, Forms, StdCtrls, ExtCtrls,
  // sgc
  sgcUICore_Theme, sgcWizard_Model, sgcWizard_VCL;
```

The handlers of the four combo boxes:

```
procedure TfrmWizardThemes.FormCreate(Sender: TObject);
begin
  { the demo opens with the dark theme, the item selected in cboTheme }
  cboThemeChange(nil);
end;

procedure TfrmWizardThemes.cboThemeChange(Sender: TObject);
begin
  if cboTheme.ItemIndex = 1 then
    sgcUISetApplicationTheme(utDark, True)
  else
    sgcUISetApplicationTheme(utLight, True);
end;

procedure TfrmWizardThemes.cboStyleChange(Sender: TObject);
begin
  if cboStyle.ItemIndex = 1 then
    sgcUISetApplicationThemePreset(CS_UI_THEME_FLUENT)
  else
    sgcUISetApplicationThemePreset(CS_UI_THEME_CLASSIC);
end;

procedure TfrmWizardThemes.cboAccentChange(Sender: TObject);
begin
  { AccentColor is an ARGB value: $AARRGGBB, the alpha first }
  case cboAccent.ItemIndex of
    1:
      Wizard.AccentColor := $FF7C3AED; // purple
    2:
      Wizard.AccentColor := $FF16A34A; // green
    3:
      Wizard.AccentColor := $FFEA580C; // orange
  else
    Wizard.AccentColor := 0; // the accent of the theme
  end;
end;

procedure TfrmWizardThemes.cboTransitionChange(Sender: TObject);
begin
  { the animation is turned off when Windows is set to reduce the motion
    (ReduceMotion wrmSystem, the default) }
  case cboTransition.ItemIndex of
    1:
      Wizard.Transition := wtSlide;
    2:
      Wizard.Transition := wtFade;
    3:
      Wizard.Transition := wtSlideFade;
  else
```

```
    Wizard.Transition := wtNone;
  end;
end;
```

How it works

1. `sgcUISetApplicationTheme` sets the theme of the whole application. Every sgcUI control whose `Theme` is `utDefault`, this wizard included, repaints in the new theme. The second parameter, True, also themes the standard VCL controls of every form, which is why the top panel and its combo boxes turn dark too.
2. The wizard themes the controls on its own pages as well: the labels, the edits and the check box of the Profile page follow every change.
3. `sgcUISetApplicationThemePreset` switches between the Classic style and the Fluent style, the look of Windows 11. Light or dark is kept.
4. `AccentColor` belongs to this wizard only. It colours the active step, the completed connectors and the Next button; 0 gives the accent back to the theme.
5. `Transition` takes effect on the next page change. Click Next and Back to see the slide go in both directions.

Try this

- Set `Wizard.Theme` to `utLight` in the Object Inspector and run again. The wizard stays light, whatever the Theme combo box says, while the top panel follows the application.
- Set `TransitionDuration` to 600 to slow the animation down.
- Set `ReduceMotion` to `wrmReduce`. The pages switch at once, whatever `Transition` says.

4.5 Demo 05: Workspace Setup

Folder `05.WizardSetupApp`, project `WizardSetupApp.dpr`.

What you will build

A small onboarding application that puts the previous demos together. Three pages ask for the account, the workspace and the plan, each page is validated before the user can move on, and a fourth page, the summary, is generated by the wizard: it lists every answer with an Edit link per page and a computed monthly price. The Finish button reads “Create workspace”.

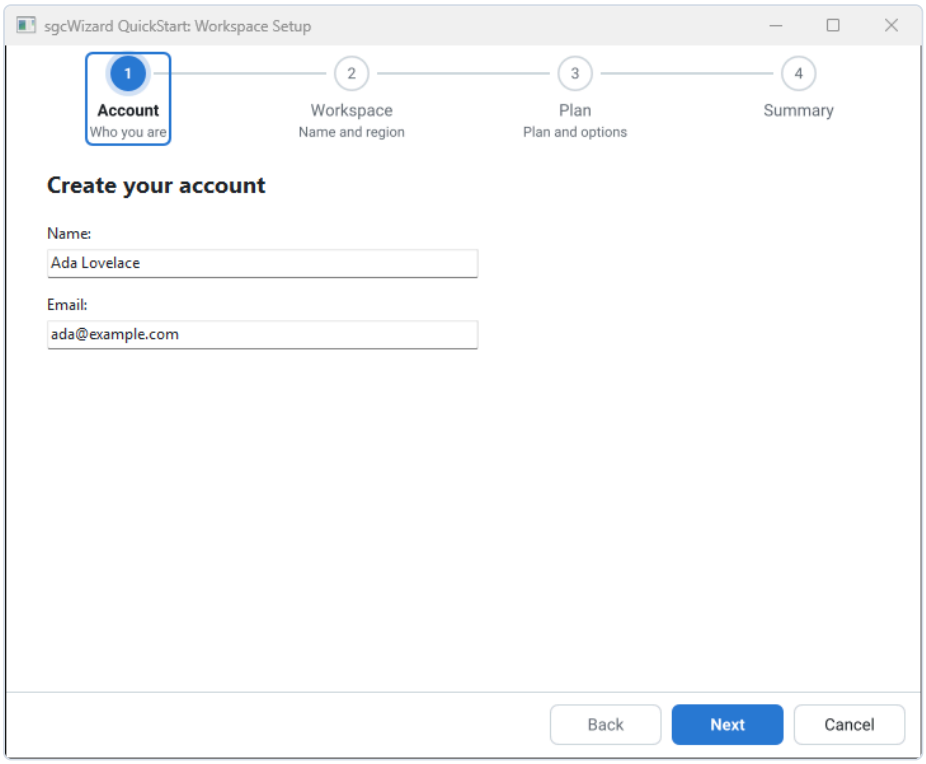


Figure 4.5. Demo 05 on its first page. The fourth step, Summary, was added by the wizard because `ShowSummary` is True.

Components on the form

Component	Class	Properties you set
`Wizard`	`TsgcVCLWizard`	`ShowSummary = True`, `SummaryTitle = 'Check your new workspace'`, `FinishCaption = 'Create workspace'`, `Transition = wtSlide`, `OnBeforePageChange`, `OnGetSummaryItems`, `OnFinish`, `OnCancel`.
`pgAccount`	`TsgcVCLWizardPage`	`Caption = 'Account'`, `Description = 'Who you are'`. The labels `lblName` and `lblEmail` with `FocusControl` set to `edtName` and `edtEmail`, and the error label `lblAccountError`.
`pgWorkspace`	`TsgcVCLWizardPage`	`Description = 'Name and region'`. The edit `edtWorkspace`, the combo boxes `cboRegion` and `cboTeam`, each with a label whose `FocusControl` points at it, and `lblWorkspaceError`.
`pgPlan`	`TsgcVCLWizardPage`	`Description = 'Plan and options'`. The combo box `cboPlan` (Free, Team, Business) with its label, and the check boxes `chkBackups` (checked) and `chkWeekly`.

The labels are the key to the summary. In the form file:

```
      object lblName: TLabel
        // ...
        Caption = 'Name:'
        FocusControl = edtName
      end
```

The code

The `uses` clause adds `sgcWizard_Model` for `TsgcWizardSummaryItems`. `FormCreate` fills in sample answers, so you can click Next straight away:

```
procedure TfrmWizardSetupApp.FormCreate(Sender: TObject);
begin
  { sample answers; clear the edits to see the validation at work }
  edtName.Text := 'Ada Lovelace';
  edtEmail.Text := 'ada@example.com';
  edtWorkspace.Text := 'Analytical Engines';
end;
```

The validation is split in two: `DoValidate` knows the rules of each page, and `OnBeforePageChange` applies them to the page being left:

```
function TfrmWizardSetupApp.DoValidate(aPage: TsgcVCLWizardPage;
  out aError: string): Boolean;
var
  vEmail: string;
begin
  aError := '';
  if aPage = pgAccount then
  begin
    vEmail := Trim(edtEmail.Text);
    if Trim(edtName.Text) = '' then
      aError := 'Enter your name.'
    else if (Pos('@', vEmail) < 2) or (Pos('.', vEmail) = 0) then
      aError := 'Enter a valid email, for example name@example.com.';
  end
  else if aPage = pgWorkspace then
  begin
    if Trim(edtWorkspace.Text) = '' then
      aError := 'Give the workspace a name.';
  end;
  Result := aError = '';
end;

procedure TfrmWizardSetupApp.WizardBeforePageChange(Sender: TObject;
  aFromIndex, aToIndex: Integer; var aAllow: Boolean);
var
  vError: string;
begin
  { Back is always allowed; a move forward checks the page being left }
  if aToIndex < aFromIndex then
    Exit;
  aAllow := DoValidate(Wizard.Pages[aFromIndex], vError);

  { the wizard paints the labels of its pages with the theme text colour,
    so the red is set in code }
  lblAccountError.Font.Color := clRed;
  lblWorkspaceError.Font.Color := clRed;
  if Wizard.Pages[aFromIndex] = pgAccount then
```

```
    lblAccountError.Caption := vError
  else if Wizard.Pages[aFromIndex] = pgWorkspace then
    lblWorkspaceError.Caption := vError;
end;
```

The summary adds one computed line to the section of the Plan page, and Finish creates the workspace:

```
procedure TfrmWizardSetupApp.WizardGetSummaryItems(Sender: TObject;
  aPage: TObject; aItems: TsgcWizardSummaryItems);
var
  vPrice: string;
begin
  { aItems already holds the answers the wizard found on aPage: add,
    change or delete lines here }
  if aPage = pgPlan then
  begin
    case cboPlan.ItemIndex of
      1:
        vPrice := '9 EUR per user';
      2:
        vPrice := '19 EUR per user';
    else
      vPrice := 'Free';
    end;
    aItems.Add('Monthly price', vPrice);
  end;
end;

procedure TfrmWizardSetupApp.WizardFinish(Sender: TObject);
begin
  ShowMessage(Format('Workspace "%s" created for %s (%s plan, %s).',
    [Trim(edtWorkspace.Text), Trim(edtName.Text), cboPlan.Text,
    cboRegion.Text]));
  Close;
end;
```

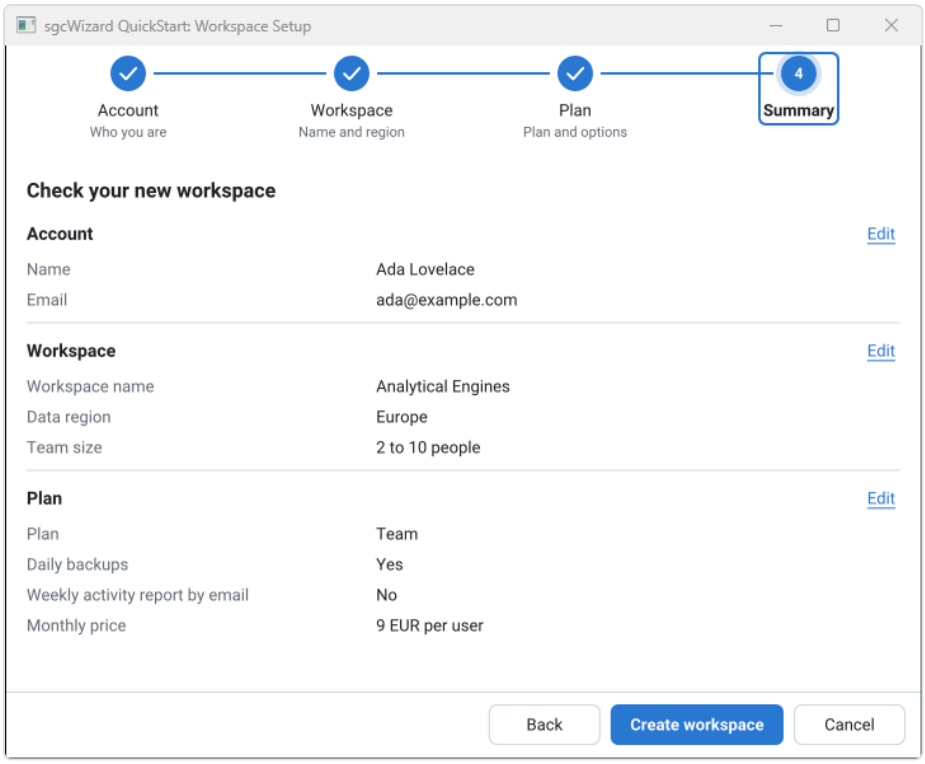


Figure 4.6. The summary page after three clicks on Next. The captions come from the labels (without the colon), the check boxes show Yes or No, and the last line comes from `OnGetSummaryItems`.

How it works

1. Because `ShowSummary` is True, the wizard adds its own page after `pgPlan`, and a fourth step named Summary. The Plan page is no longer the last one, so its Next button reads Next, and the Summary page shows the `FinishCaption`.
2. Each click on Next fires `OnBeforePageChange`. `Wizard.Pages[aFromIndex]` is the page being left, and `DoValidate` checks it. The move from the Plan page to the summary is validated too; the Plan page simply has no rules.
3. When the summary is shown, the wizard walks the visited pages. For every input control with a label pointing at it, or with a caption of its own, it adds a line: the label text and the value.
4. For each page it then calls `OnGetSummaryItems` with the lines found. The demo adds the monthly price to the Plan section.
5. An Edit link goes back to its page. After a change there, Next returns straight to the summary.
6. Create workspace fires `OnFinish`, which shows what would be created and closes the form.

Try this

- Set `SummaryCaption` to `Review`. The fourth step is renamed.
- In `OnGetSummaryItems`, remove the weekly report line from the Plan section with `aItems.IndexOf` and `aItems.Delete`.
- Set `StepperPosition` to `wspLeft` and `Theme` to `utDark` for a dark onboarding window with a vertical list of steps.

5. Main components

The SGC Wizard palette page is short: a wizard and a stepper. This chapter introduces both, the page class you add with the component editor, and the members you will want once the basics are in place.

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

Reference table

Class	Palette page	What it is for	Key members
`TsgcVCLWizard`	SGC Wizard	The VCL wizard: pages, stepper band and button bar	`ActivePageIndex`, `StepperPosition`, `ShowSummary`, `Theme`, `Transition`, `Next`, `Back`, `GoToPage`, `OnBeforePageChange`, `OnPageChanged`, `OnFinish`, `OnCancel`
`TsgcVCLWizardPage`	None, added with **Add Page**	One page, one step	`Caption`, `Description`, `StepGlyph`, `Skip`, `PageIndex`
`TsgcVCLStepper`	SGC Wizard	A step indicator on its own	`Steps`, `Orientation`, `ActiveIndex`, `AutoStates`, `Clickable`, `OnStepClick`, `OnActiveChange`
`TsgcFMXWizard`, `TsgcFMXWizardPage`, `TsgcFMXStepper`	SGC Wizard (FireMonkey)	The FireMonkey twins	The same as the VCL classes
`TsgcWizardSummaryItems`	None	The lines of one section of the summary, in `OnGetSummaryItems`	`Add`, `IndexOf`, `Delete`, `Count`, `Captions`, `Values`

The wizard

`TsgcVCLWizard` in unit `sgcWizard_VCL` is the component you use in almost every case. Its published properties fall into a few groups:

Group	Properties
Pages	`ActivePageIndex`
Steps	`StepperPosition` (`wspTop` , `wspLeft` , `wspNone`)
Buttons	`BackCaption` , `NextCaption` , `FinishCaption` , `CancelCaption` , `ShowCancel`
Summary	`ShowSummary` , `SummaryTitle` , `SummaryCaption`
Look	`Theme` , `AccentColor` , `Transition` , `TransitionDuration` , `ReduceMotion` , `BiDiMode`
Accessibility	`AccessibleName` , `AccessibleDescription`
Events	`OnBeforePageChange` , `OnPageChanged` , `OnFinish` , `OnCancel` , `OnTransitionStart` , `OnTransitionEnd` , `OnGetSummaryItems` , `OnSummaryEdit`

At run time you also have `ActivePage` , `PageCount` , `Pages[]` , the navigation methods `Next` , `Back` , `GoToPage` , `Finish` and `Cancel` , `AddPage` to create pages in code, and `SummaryPage` , the generated summary page (nil while `ShowSummary` is False). With `BiDiMode` set to right to left, the stepper band and the button bar are mirrored and a `wspLeft` band docks on the right.

The pages

`TsgcVCLWizardPage` is a container with four properties of its own, `Caption` , `Description` , `StepGlyph` and `Skip` , plus the read only `PageIndex` and `Wizard` . It also publishes `Font` , `ParentFont` , `Hint` , `PopupMenu` and the `OnEnter` , `OnExit` and `OnResize` events. Its size and position are managed by the wizard: you never align or move a page yourself.

The stepper on its own

`TsgcVCLStepper` is the step indicator of the wizard as a separate control, for flows that are not page based: the progress of a long running job, the status of an order, a checkout spread over several forms. Its steps are a collection, `Steps` , that you fill in the Object Inspector or in code; each step has a `Caption` , a `Description` , a `Glyph` and a `State` . `Orientation` chooses horizontal or vertical. With `AutoStates` True, the default, setting `ActiveIndex` marks the steps before it as completed. A step can also be given a state of its own, such as an error:

```
procedure TfrmSnippets.BuildStepper;
begin
  Stepper := TsgcVCLStepper.Create(Self);
  Stepper.Parent := Self;
  Stepper.Align := alTop;
  Stepper.Steps.AddStep('Upload', 'Choose the file');
  Stepper.Steps.AddStep('Convert', 'To PDF');
  Stepper.Steps.AddStep('Download');
  { with AutoStates (the default) the steps before ActiveIndex are drawn
    as completed }
  Stepper.ActiveIndex := 1;
  { a step can also be given a state of its own }
  Stepper.Steps.Items[1].State := wssError;
end;
```

The states are `wssPending`, `wssActive`, `wssCompleted`, `wssError` and `wssSkipped`, declared in `sgcWizard_Model`, so that unit goes into the `uses` clause next to `sgcWizard_VCL`. `OnStepClick` tells you which step the user clicked and lets you refuse it. The `Stepper` demo, listed in chapter 7, shows both orientations.

Beyond the basics

- **OnSummaryEdit** fires when the user clicks an Edit link of the summary, and can refuse it.
- **OnTransitionStart** and **OnTransitionEnd** bracket the animation of a page change, and `FinishTransition` ends a running animation at once.
- **Headless output.** The unit `sgcWizard_Render` has `sgcWizardSaveToPNG` and `sgcWizardSaveToSVG`, which write an image of a stepper model to a file or a stream without any window, for example for a report or a web page.
- **Accessibility.** The wizard, its steps and its buttons are exposed to screen readers. `AccessibleName` and `AccessibleDescription` set what a screen reader announces for the wizard.

6. Common problems

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

Designing the wizard

There is no page component on the palette

What you see. The SGC Wizard page shows the wizard and the stepper, but nothing to drop as a page.

Why. `TsgcVCLWizardPage` is registered as a class, not as a palette component, because a page can only live on a wizard.

Fix. Right click the wizard, or one of its pages, and choose **Add Page**. The new page gets a default name and caption; rename both in the Object Inspector.

EsgcWizardError: a page can only be placed on a wizard

What you see. Moving a page onto a panel, or setting the `Parent` of a page in code, raises `EsgcWizardError` with the message "A TsgcVCLWizardPage can only be placed on a TsgcVCLWizard".

Why. A page must be a direct child of a wizard.

Fix. Create pages with Add Page at design time or `AddPage` in code. To group controls on a page, put a `TPanel` or a frame on the page.

The wizard opens on the wrong page

What you see. The application starts on the second or the last page instead of the first.

Why. **Add Page**, **Next Page** and **Previous Page** change `ActivePageIndex` in the designer, to show you the page you are working on, and that value is saved in the form file.

Fix. Before you save, choose Previous Page until the first page is shown, or set `ActivePageIndex` back to 0 in the Object Inspector. Demo 02 opens on its second page on purpose, by setting `ActivePageIndex = 1`.

Navigation and events

OnPageChanged does not fire when the form opens

What you see. Code in `OnPageChanged` runs after every click on Next or Back, but not for the first page shown, so whatever it fills is empty at start.

Why. Setting `ActivePageIndex`, in the form file or in code, switches the page without events. Only the navigation methods fire them.

Fix. Call the handler once yourself in `FormCreate`, as demo 02 does with `WizardPageChanged(Wizard)`.

Next does nothing

What you see. A click on Next leaves the wizard on the same page.

Why. `OnBeforePageChange` set `aAllow` to False, which is how the validation refuses a move. If a handler forgets to show a message, the refusal looks like a dead button.

Fix. Always tell the user why, as demos 03 and 05 do with a red label. Check that every branch of the handler that refuses also explains, and that a move backwards is let through with `if aToIndex < aFromIndex then Exit`.

Finish does not close the form

What you see. Clicking Finish or Cancel does nothing visible.

Why. The wizard never closes anything. It only fires `OnFinish` or `OnCancel`.

Fix. Call `Close` in both handlers, or set `ModalResult` when the form is shown with `ShowModal`, as in Finish and Cancel. Remember that Escape fires `OnCancel` even when `ShowCancel` is False.

Enter does not move to the next page

What you see. Enter works on most pages, but not on one of them.

Why. A control that wants Enter keeps it: a `TMemo`, a focused button, or a `TButton` on the form with `Default` set to True.

Fix. Remove `Default` from buttons on the wizard form, or accept that Enter belongs to the memo on that page; the Next button still works.

A click on a step does not change the page

What you see. Clicking a step of the band does nothing.

Why. Only completed steps can be clicked, so the user cannot jump ahead past the validation. A click also goes through `OnBeforePageChange`, which may refuse it.

Fix. Nothing to fix for steps ahead of the active one. To jump ahead from code, call `GoToPage`.

PageCount is one more than the pages you designed

What you see. With `ShowSummary` True, `Wizard.PageCount` returns 4 for a wizard with three designed pages.

Why. The generated summary is a real page of the wizard, after the last designed page.

Fix. Compare pages, not numbers: `Wizard.ActivePage = Wizard.SummaryPage`, or `aFromIndex = pgPlan.PageIndex`. On Finish, `aToIndex` is still `PageCount`.

A field is missing from the summary

What you see. The summary lists some answers of a page but not all of them.

Why. The summary needs a caption for every value. It takes it from a `TLabel` whose `FocusControl` points at the control, or from the control's own caption.

Fix. Set `FocusControl` on the label of the edit or the combo box, as in demo 05. For anything the summary cannot find, add a line in `OnGetSummaryItems`.

Colours and themes

The red error label is not red

What you see. A label set to `clRed` in the Object Inspector shows in the normal text colour at run time.

Why. The wizard themes the standard controls on its pages, when it loads and on every theme change. That sets the font colour of every label to the text colour of the theme, which keeps a dark wizard readable but also overwrites your red.

Fix. Set the colour in code, right before you write the message, as demos 03 and 05 do:

```
lblNameError.Font.Color := clRed;
lblAddressError.Font.Color := clRed;
```

The wizard ignores the application theme

What you see. `sgcUISetApplicationTheme` or `sgcUISetApplicationThemePreset` changes every other sgcUI control, but one wizard keeps its old look.

Why. Its `Theme` property is set to `utLight` or `utDark`. A wizard with an explicit theme ignores later changes of the application theme and style; only `utDefault` follows them.

Fix. Set `Theme` back to `utDefault` and drive the look through the application theme, as demo 04 does. Use an explicit theme only when this one wizard must look different on purpose.

The pages switch without animation

What you see. `Transition` is set, but the pages switch at once.

Why. `ReduceMotion` is `wrmSystem`, the default, and Windows is set to show no animations; or `TransitionDuration` is 0. Setting `ActivePageIndex` in code never animates, only the navigation does.

Fix. Respect the user's Windows setting if you can. To animate anyway, set `ReduceMotion` to `wrmAnimate`.

Running and testing

A test program asks for administrator rights

What you see. Windows asks for elevation before it starts your wizard test program.

Why. Windows treats an executable without a manifest whose name contains words such as setup, install or update as an installer.

Fix. Keep the application manifest that Delphi adds to new projects (**Project** > **Options** > **Application** > **Manifest**), with the execution level `asInvoker`, or choose a name such as `Onboarding.exe`.

7. Where next

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

The other Delphi demos

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

Demo	What it shows
`Onboarding`	An account wizard with a validation that vetoes Next, a page that can be skipped, a built in summary page with Edit links and a light and dark theme switch. The natural next step after demo 05.
`OnboardingFMX`	The same account wizard built with `TsgcFMXWizard` for FireMonkey.
`Stepper`	A horizontal and a vertical `TsgcVCLStepper` driven by buttons that advance, go back, mark an error, skip a step and reset, with a light and dark theme switch.

Help topics

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

Topic	What it covers
Introduction	What sgcWizard does and where to start.
Architecture	The stepper model, the renderer, the VCL and FireMonkey controls, the stepper band and the button bar.
Platforms	The Delphi versions, the target platforms and the packages.
Navigation, Validation, Finish and Cancel, Keyboard	Moving between the pages, skipped pages, refusing a move and ending the wizard.
Step states, Stepper layout, Button captions	The look of the steps and the buttons.
Themes, Page transitions, Summary page	Light and dark, the accent, the animations and the generated summary.
Design time, Headless rendering	The component editor, and PNG and SVG output without a window.
Troubleshooting	More problems and their causes.
One topic per class	`TsgcVCLWizard`, `TsgcVCLWizardPage`, `TsgcVCLStepper`, `TsgcWizardSummaryItems` and the others, with every property, its type and its default.

On the website

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

Enjoy guiding your users, one step at a time.

---

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