sgcNavigation

Getting Started

Give a Delphi application its side menu, document tabs, breadcrumb and toasts, one small demo at a time

About this guide

This guide is for Delphi developers who know how to build a VCL form and want to give their application a modern navigation shell with sgcNavigation: a side menu on the left, tabs for the open documents, a breadcrumb that tells the user where they are, and small pop-up notifications in the corner. You do not need to know anything about sgcNavigation or the rest of sgcUI in advance.

The guide follows the five QuickStart demos. Each demo adds one idea to the previous one, from a side menu with five items and one event handler to a small mail client that combines a folder tree, a breadcrumb, message tabs and a toast. 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 around, then read its section and try the suggested changes. Chapters 1 to 3 explain what sgcNavigation 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\Navigation\Delphi\00.QuickStart` folder of your sgcUI installation. Each demo has its own subfolder, from `01.NavHello` to `05.NavMailDesk`, 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 a Spanish locale; numbers and dates in the demos follow the Windows regional settings of the machine that runs them.

Contents

1. What sgcNavigation is

The idea in one minute
Your first side menu needs one event handler
What you need
Part of sgcUI
Coming from panels, buttons and page controls

2. Install and the palette

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

3. Core concepts

The bar shows a flat list of items
Three modes: sidebar, accordion, tree
Feeding the items
Selection and the events of the bar
The bar does not own your pages
Tabs show the Control of the active tab
The breadcrumb follows the bar
Toasts and notifications
Themes, accent colour and the rail
Main thread only
Summary: who does what

4. The QuickStart demos

4.1 Demo 01: Hello Navigation
4.2 Demo 02: Menu Builder
4.3 Demo 03: Document Tabs
4.4 Demo 04: Appearance
4.5 Demo 05: Mail Desk

5. Main components

Reference table
The navigation bar
Tabs and the notebook
The breadcrumb
Notifications
Beyond the basics

6. Common problems

The navigation bar
Tabs, breadcrumb and toasts
Editions and compiling

7. Where next

The other Delphi demos
Help topics
On the website
Getting help

1. What sgcNavigation is

sgcNavigation gives a Delphi application its navigation shell, for the VCL and for FireMonkey. It is the frame around your pages: the side menu that lists the parts of the application, the tab strip that holds the open documents, the breadcrumb that shows where the user is, and the small notifications that pop up in a corner when something happens. Your own forms, grids and editors live inside that frame.

You drop a `TsgcVCLNavigationBar` on a form, type its items in the Object Inspector or add them in code, and react when the user picks one. Next to it you can drop a `TsgcVCLNavigationTabs` for documents, a `TsgcVCLNavigationBreadcrumb` that follows the bar, and a `TsgcVCLToastManager` for notifications. Every control draws itself, in a light, dark or high contrast theme, sharp at any screen scale.

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

Most business applications have the same layout. A narrow column on the left lists the areas of the program: Home, Inbox, Calendar, Contacts, Settings. A click on an entry shows its page on the right. When the program opens documents, a row of tabs above the page lets the user switch between them, close them and open new ones.

sgcNavigation splits that layout into a few independent controls:

- **The navigation bar** is the column on the left. It shows a list of *items*, each with a caption, a small icon or glyph, an optional badge with a counter, and a hint. It can be a flat list, an accordion of groups or a tree, it can fold into a narrow rail of icons, and it can show a search box that filters the items.
- **The tab strip** shows one tab per open page or document. Each tab can carry a control, the page, and the strip shows the page of the active tab and hides the others.
- **The breadcrumb** shows the path of the current location, such as *Inbox* > *Customers*, and lets the user jump back up.
- **The toast manager** pops up short messages, such as “New message”, that close by themselves after a few seconds.

None of them knows about the others unless you connect them, and none of them owns your pages. The bar tells you which item the user picked; you decide what to show. That is the whole idea, and the rest of the guide fills in the details.

Your first side menu needs one event handler

The first QuickStart demo is the smallest navigation application there is. The form holds a navigation bar and a label. The five items of the bar are typed in the Object Inspector and stored in the form file like this:

```
  object Nav: TsgcVCLNavigationBar
    Items = <
      item
        Caption = 'Home'
        Glyph = 'H'
      end
      item
        Caption = 'Inbox'
        Glyph = 'I'
      end
      // ...
      item
        Caption = 'Settings'
        Glyph = 'S'
      end>
    ExpandedWidth = 220
    SelectedIndex = 0
    OnItemClick = NavItemClick
  end
```

The only code is the `OnItemClick` event of the bar. The bar passes the item that was clicked, and the label shows its caption:

```
procedure TfrmNavHello.NavItemClick(Sender: TObject;
  aItem: TsgcNavigationItem);
begin
  { aItem is the TsgcNavigationItem that was clicked: one entry of
    Nav.Items, with the Caption set in the Object Inspector. A real
    application would show the page of that item here. }
  lblPage.Caption := aItem.Caption;
end;
```

That is all. You did not write any drawing code, any mouse handling or any code for the selection highlight, the hot row under the mouse or the keyboard. The bar does all of that by itself. In a real application the line in the handler would show a page instead of setting a caption.

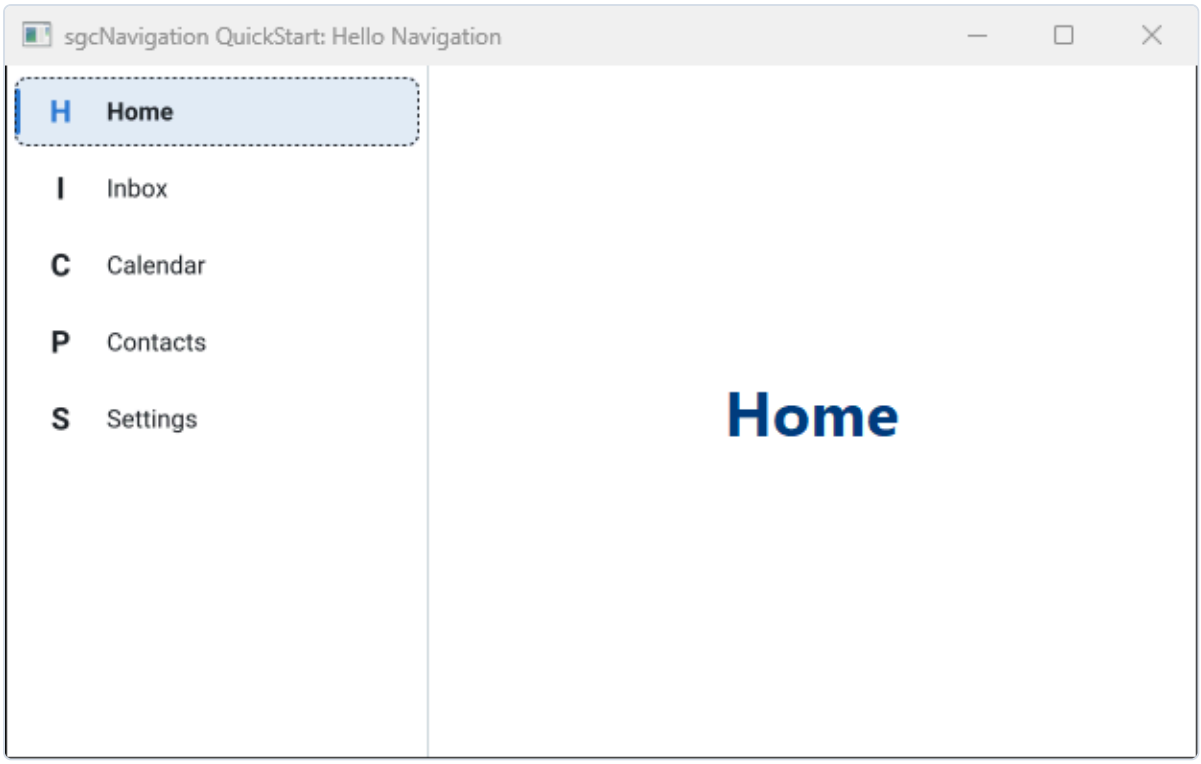


Figure 1.1. The result of demo 01: five items, Home selected, and the caption of the clicked item on the right.

What you need

- **Delphi.** The packages in the `Packages` folder cover Delphi 7, 2007, 2009, 2010, XE to XE8, 10 Seattle to 10.4 Sydney, 11, 12 and 13.
- **VCL.** The VCL controls (`TsgcVCLNavigationBar` , `TsgcVCLNavigationTabs` , `TsgcVCLNavigationBreadcrumb` , `TsgcVCLNavigationNotebook` , `TsgcVCLToastManager` and the rest) run on Windows, Win32 in every version and Win64 where your Delphi version can build it.
- **FireMonkey.** The FireMonkey twins, such as `TsgcFMXNavigationBar` and `TsgcFMXNavigationTabs` , have their own packages from Delphi XE6 on, for the platforms FireMonkey targets in your Delphi version.
- **No window at all.** The model and the renderer of the bar use no visual framework, so a bar can also be written to a PNG or SVG image, for example in documentation or on a server.
- **sgcUICore.** The shared foundation of the sgcUI visual products. It is installed together with sgcNavigation, you do not install it separately.

The QuickStart demos are VCL applications and need nothing else: no images, no database and no network. The icons of the items are short glyph letters drawn by the bar itself.

Part of sgcUI

sgcNavigation 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 two things for you:

- A navigation bar 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, and one call, `sgcUISetApplicationTheme` , changes the theme of every sgcUI

control that keeps `Theme = utDefault`. Demo 04 shows it.

- sgcNavigation is meant to frame the other products: a navigation bar on the left, and on the right an sgcDashboard dashboard, an sgcGrid grid or a page with an sgcToolbar. Two of the larger demos listed in chapter 7 host a dashboard.

Coming from panels, buttons and page controls

Many Delphi applications build their side menu by hand: a `TPanel` aligned to the left with a column of `TSpeedButton`s, a `TTreeView`, a `TButtonGroup` or a `TCategoryButtons`, and a `TPageControl` with hidden tabs for the pages. That works, but the details take time: the selection colour, the badges, the folding groups, the search box, the narrow icon mode, the dark theme and high DPI.

With sgcNavigation those details are properties. The part you keep is the one that is really yours: what each item shows. You can even keep your `TPageControl` or `TCardPanel` for the pages and let the bar switch them, as chapter 3 shows. If you build only for a recent RAD Studio, the help topic *Choosing sgcNavigation* compares sgcNavigation with the navigation controls that RAD Studio itself added to the VCL.

2. Install and the palette

This chapter gets sgcNavigation 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

sgcNavigation 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 **sgcNavigation** under Products. The installer shows one checkbox per product and enables the products your licence includes. sgcUICore is installed automatically with it.
4. Keep **Demo Samples** selected if you want the demos of this guide.
5. The installer copies the files, compiles the packages on your machine for each selected Delphi version and installs the design time packages.

Everything goes into one folder under `Documents\esegece`, 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 sgcNavigation units without any setting on your side.

The packages

You only need this section if you build the packages yourself. sgcNavigation 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
`sgcNavigation<V>`	Runtime	The models, the renderers, the notifications and the headless output. No VCL, no FireMonkey. Requires `sgcUICore<V>`.
`sgcNavigationVCL<V>`	Runtime	The VCL controls: the bar, the tabs, the breadcrumb, the notebook, the toast manager, the inbox and the rest. Requires `sgcUICoreVCL<V>`.
`sgcNavigationFMX<V>`	Runtime, XE6 and later	The FireMonkey controls.
`dclsgcNavigation<V>`	Design time	Registers the VCL controls and the non visual components on the palette, with the component editor of the notebook.
`dclsgcNavigationFMX<V>`	Design time, XE6 and later	Registers the FireMonkey controls.

Build them in dependency order: the sgcUICore packages first, then `sgcNavigation`, then `sgcNavigationVCL`, then `dclsgcNavigation`. 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 Navigation**. In a VCL project it shows these components:

Component	Visual	What it does
`TsgcVCLNavigationBar`	yes	The side menu: items with glyphs or images, badges, groups, a tree, a search box and a folding rail.
`TsgcVCLNavigationTabs`	yes	A tab strip for documents or pages, with close, new and list buttons.
`TsgcVCLNavigationBreadcrumb`	yes	The path of the current location, which can follow a navigation bar.
`TsgcVCLNavigationNotebook`	yes	A page control on the tab strip, with pages you design in the form.
`TsgcVCLToastManager`	no	Pops up short notifications in a corner of the form or the screen.
`TsgcVCLNotificationInbox`	yes	A list of the notifications of a notification model, with read and unread state.
`TsgcVCLNotificationPrefsEditor`	yes	Lets the user switch the notification channels per group on and off.
`TsgcNotificationModel`	no	The list of notifications shared by the toast manager and the inbox.
`TsgcNotificationPreferences`	no	The per group settings used by the toast manager and the inbox.
`TsgcVCLCommandPalette`, `TsgcVCLActionListCommandSource`	yes, no	A keyboard command palette with fuzzy search, and a source that feeds it the actions of a `TActionList`. Not in the Community edition.
`TsgcVCLDockManager`	yes	An IDE style workspace with docked, tabbed, auto hidden and floating panels. Not in the Community edition.

In a FireMonkey project the same page shows the FireMonkey twins, such as `TsgcFMXNavigationBar`, instead of the VCL controls, together with the same non visual models. The QuickStart demos use only the bar, the tabs, the breadcrumb and the toast manager.

Help at design time

- **The collection editors.** The `Items` of a navigation bar and the `Tabs` of a tab strip are collections. Select the control, click the ellipsis button of `Items` or `Tabs` in the Object Inspector, and the collection editor opens with one entry per item or tab. Select an entry to set its `Caption`, `Glyph`, `Badge` and the rest in the Object Inspector.
- **The notebook editor.** Right click a `TsgcVCLNavigationNotebook` to find **New Page**, **Delete Page**, **Next Page** and **Previous Page**. Pages are created with **New Page**, never from the palette, and you drop your controls on them like on the sheets of a `TPageControl`.
- **A live preview.** The controls paint in the form designer with the items, badges, tabs and theme you set, so you see the result before you run.

Editions

sgcNavigation is built from the same source for every edition. The Registered and Trial editions include everything; the trial is meant for evaluation. The free Community edition is compiled without two groups of features, controlled by two defines in `sgcVer.inc`:

- `SGC_NAVIGATION_ADVANCED`: the command palette, the drag reorder of tabs, pinned tabs, tear off, the edit mode of the breadcrumb path and the docking manager.
- `SGC_NAVIGATION_ACTIONS`: the `Action` property of the items, which links an item to a `TAction`.

Everything else is in every edition: the navigation bar in all three modes with search, badges, footer items and the folding rail, the tab strip, the breadcrumb that follows the bar, the notebook, the toasts and the notification inbox.

> **Note.** The QuickStart demos deliberately avoid both groups, so they compile and run with every edition. The `Accordion`, `DockingShell` and `NavigationShell` demos of chapter 7 show some of the advanced features.

Where the demos are

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

Folder	Project	What it shows
01.NavHello	`NavHello.dpr`	The smallest application: a bar with five items and one event handler
02.NavMenuBuilder	`NavMenuBuilder.dpr`	The menu built in code: accordion groups, badges, a disabled item, a footer item, search and the menu button
03.NavDocTabs	`NavDocTabs.dpr`	A document tab strip: a memo per tab, new and close buttons, modified tabs
04.NavAppearance	`NavAppearance.dpr`	Theme, style and accent colour for the whole application, and the folded rail
05.NavMailDesk	`NavMailDesk.dpr`	A small mail client: folder tree with unread badges, breadcrumb, message tabs and a toast

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

Running your first demo

1. In Delphi, choose **File** > **Open Project** and open `Demos\Navigation\Delphi\00.QuickStart\01.NavHello\NavHello.dpr`.
2. Press **F9**. The window opens with Home selected in the bar.
3. Click the items. The label on the right shows the caption of each one. Try the keyboard too: click the bar once, then use the arrow keys and Enter.
4. Back in the IDE, open the form, select `Nav` and click the ellipsis button of `Items` in the Object Inspector. Those five entries are the whole menu.

3. Core concepts

Ten ideas explain almost everything the navigation controls do. Read them once now, and come back to them when a demo surprises you.

The bar shows a flat list of items

The content of a navigation bar is its `Items` property, a collection of `TsgcNavigationItem`. You usually edit it in the Object Inspector, and it is saved in the form file. The collection is always **flat**: one list, read from top to bottom. Groups and trees are made with properties of the items, not with nested lists. These are the item properties you will set most often:

Property	Meaning
`Caption`	The text of the row.
`Glyph`	A short text or symbol drawn in the icon slot when there is no image, such as `'H'` or `'*'`. The QuickStart demos use glyph letters.
`ImageIndex`	An image of the `Images` image list of the bar, -1 for none.
`Badge`	A small counter drawn on the right, such as `'12'`. An empty string hides it.
`Hint`	The tooltip of the row, shown when `ShowHint` is on.
`IsGroup`	Makes the item a group header. It owns the items that follow it, up to the next header.
`Depth`	The level in a tree, 0 for the top. An item followed by deeper items is their parent.
`Expanded`	Whether a group header or a tree parent shows its children. `True` by default.
`Placement`	`nipTop` (the default) or `nipFooter`, which pins a top level item at the bottom of the bar, like Settings in the demos.
`Enabled`, `Visible`	A disabled item is painted greyed and ignores the mouse and the keyboard. A hidden item is not shown at all.
`Tag`	A number for your own use, for example the index of the page of the item.
`OnClick`	An event of the item itself, as an alternative to the `OnItemClick` event of the bar.

The units are `sgcNavigation_Model` for the items and the enumerations such as `nbmAccordion` and `nipFooter`, and `sgcNavigation_VCL` for the bar. Add both to the `uses` clause of a form that works with items in code.

Three modes: sidebar, accordion, tree

The `Mode` property decides how the bar reads the flat list:

- **`nbmSidebar`**, the default. A plain list. Group headers are static labels that separate the sections.
- **`nbmAccordion`**. A click on a group header folds or unfolds the items below it. With `AllowMultiExpand = False`, the default, opening a group closes the others; with `True` several groups stay open. Demo 02 uses it.

- **`nbmTree`**. A multi level tree made with `Depth`: an item with `Depth = 1` right after an item with `Depth = 0` is its child. A parent folds and unfolds with its expander arrow, and a click on its caption selects it like any other item. Demo 05 uses it for mail folders.

In every mode, `ShowSearch = True` adds a search box at the top. Typing filters the items by caption and keeps the parents of every match visible. `FilterMode` chooses between a plain substring match (`nfmContains`, the default) and a fuzzy match (`nfmFuzzy`), where "ivc" finds "Invoices". The `Filter` property holds the text, so you can also filter from code.

Feeding the items

A fixed menu is typed at design time, as in demo 01. A menu that depends on the modules, the user rights or a configuration file is built in code, as in demo 02. `Items.AddItem(Caption, Glyph, IsGroup)` adds an item and returns it, so you can set its other properties on the result. Wrap the changes in `BeginUpdate` and `EndUpdate`, so the bar is laid out and painted once at the end instead of after every item. This snippet builds the folder tree of demo 05 in code:

```
procedure BuildTree(Nav: TsgcVCLNavigationBar);
var
  oItem: TsgcNavigationItem;
begin
  Nav.Mode := nbmTree;
  Nav.Items.BeginUpdate;
  try
    Nav.Items.Clear;
    Nav.Items.AddItem('Inbox', 'I');
    oItem := Nav.Items.AddItem('Customers', 'C');
    oItem.Depth := 1; // a child of Inbox, the item above it
    oItem := Nav.Items.AddItem('Suppliers', 'U');
    oItem.Depth := 1;
    Nav.Items.AddItem('Sent', 'S');
    { Inbox starts folded: Expanded goes on the parent }
    Nav.Items[0].Expanded := False;
  finally
    Nav.Items.EndUpdate;
  end;
end;
```

Changing an item later, such as its `Badge` when new mail arrives, needs no bracket and no refresh call. A change only schedules a new frame; the bar repaints once, shortly after, however many properties you changed in a row.

Selection and the events of the bar

`SelectedIndex` is the selected item, as an index into the flat `Items` collection, or -1 when nothing is selected. Group headers count in that index, although they are never selected themselves: in demo 02, Orders has index 2 because Dashboard is 0 and the Sales header is 1. Set `SelectedIndex` in the Object Inspector or in code to choose the item that is selected when the form opens.

Two events tell you what the user did, and they answer different questions:

Event	When it fires	Use it for
`OnSelectionChange`	After `SelectedIndex` changed, by a click, the keyboard or your code. Not while the form loads.	Showing the page of the selected item. Demos 02 and 05.
`OnItemClick`	When the user clicks an enabled item, or presses Enter on it (or Space, when the bar has no search box). It receives the item, and it fires even when that item was already selected.	Running a command, or reacting to every click. Demo 01.

On a click the order is: `SelectedIndex` changes (and `OnSelectionChange` fires if it really changed), then `OnItemClick`, then the `OnClick` of the item. A click on a group header never fires either event: in the accordion it folds the group, in the sidebar it does nothing. A disabled item ignores clicks.

The bar does not own your pages

A navigation bar has no pages of its own. It tells you which item is selected, and your code shows the matching page. The simplest way is to keep the pages in a `TPageControl` (with its tabs hidden) or a `TCardPanel` and to store the page index of every item in its `Tag`, in the Object Inspector or in code. Then one line in `OnSelectionChange` switches the page:

```
procedure ShowPage(Nav: TsgcVCLNavigationBar; Pages: TPageControl);
begin
  { called from OnSelectionChange }
  if Nav.SelectedIndex >= 0 then
    Pages.ActivePageIndex := Nav.Items[Nav.SelectedIndex].Tag;
end;
```

Demo 05 uses the same idea without a page control: the index of the selected folder decides which messages go into the list.

Tabs show the Control of the active tab

A `TsgcVCLNavigationTabs` has a `Tabs` collection of `TsgcNavigationTab` and an `ActiveIndex`. Every tab has a `Caption`, an optional `Glyph` or image, and a `Control` property: the page of the tab, any control of the form. The strip shows the `Control` of the active tab and hides the controls of the other tabs, so switching pages needs no code at all. The units are `sgcNavigation_Tabs` for the tab class and the enumerations, and `sgcNavigation_Tabs_VCL` for the controls.

The strip draws the buttons, and your code decides what they do:

- `OnNewTab` fires when the user clicks the + button (`ShowNewButton`, `True` by default). The strip adds nothing by itself: you call `Tabs.Tabs.AddTab(Caption)`, which returns the new tab, and set its `Control`.
- `OnTabClose` fires when the user clicks the close button of a tab, or middle clicks it. Set `aAllow := False` to keep the tab; otherwise the strip removes the tab when the event returns. The control of the tab is not freed for you.
- `OnTabChanging` fires before another tab becomes active and can refuse it; `OnTabChange` fires after.

Three tab properties come up all the time: `Closable` (`True` by default) shows the close button, `Modified` draws a dot in the close slot of a changed document, and `Tag` is free for your own use. On the strip, `CloseButtons` decides when the close buttons show: always (`tcbAlways`, the default), on the active tab

(`tcbActive`) or on the active tab and the tab under the mouse (`tcbHover`).

The breadcrumb follows the bar

A `TsgcVCLNavigationBreadcrumb` shows a path as a row of segments. Set its `NavigationBar` property to a navigation bar, and the segments follow the selected item and its parents, such as *Inbox* > *Customers*, without any code. A click on a segment selects that item in the bar, which in turn fires the `OnSelectionChange` of the bar, so your page switching code runs as usual. Without a bar, you fill the `Segments` or set the `Path` yourself and react in `OnSegmentClick` .

Toasts and notifications

A toast is a small card that pops up in a corner, shows a title and a text, and closes by itself. Drop a `TsgcVCLToastManager` on the form and call `Show` :

```
procedure Notify(Toasts: TsgcVCLToastManager);
begin
  Toasts.Position := tpsTopRight;
  Toasts.Show('Saved', 'The invoice was saved.', nsvSuccess);
  { the fourth parameter overrides Duration for this toast, in ms }
  Toasts.Show('Disk almost full', 'Only 2 GB left on drive C.',
    nsvWarning, 10000);
end;
```

- The severity, `nsvInfo` , `nsvSuccess` , `nsvWarning` or `nsvError` , chooses the severity glyph drawn on the toast. It is declared in `sgcNavigation_Notification` .
- `Position` chooses the corner, `tpsBottomRight` by default; its values are declared in `sgcNavigation_Toast` , so a form that sets it needs that unit in its `uses` clause as well as `sgcNavigation_Toast_VCL` . `Anchor` chooses the area: the client area of the form (`tanForm` , the default) or the screen (`tanScreen`).
- `Duration` is 5000 ms by default, and the toast pauses while the mouse is over it (`PauseOnHover`). At most `MaxVisible` toasts (3 by default) show at once; the rest wait their turn.
- Each toast is a small window of its own that does not take the focus, so the user can keep typing while it shows.

For an application with a real notification centre, a `TsgcNotificationModel` keeps the list of notifications with their read state, a `TsgcVCLNotificationInbox` shows that list, and a toast manager whose `Model` points to it pops up a toast for every new unread notification. The QuickStart demos only use `Show` .

Themes, accent colour and the rail

Every sgcUI control has a `Theme` property. Its default, `utDefault` , means “follow the application theme”, so one call themes all of them at once. These three procedures of `sgcUICore_Theme` set the look of the whole application, and demo 04 calls all three:

- `sgcUISetApplicationTheme(utDark, True)` sets the theme: `utLight` , `utDark` or `utHighContrast` . The second parameter, `True` , also themes the standard VCL controls of every form, such as panels, labels and combo boxes; it needs `sgcUICore_Theme_VCL` in the `uses` clause.
- `sgcUISetApplicationThemePreset(CS_UI_THEME_FLUENT)` sets the style the themes are drawn with: `CS_UI_THEME_CLASSIC` or `CS_UI_THEME_FLUENT` , which is rounder and softer.

- `sgcUISetApplicationAccentColor(Color)` sets the accent colour of the selection, the active tab and the badges. Colours are `TsgcUIColor` values in `$AARRGGBB` form, easiest to build with `sgcUIARGB(Alpha, Red, Green, Blue)`. 0 picks the accent automatically.

To pin one control to its own look, set its own `Theme` or `AccentColor`:

```
procedure PinLook(Nav: TsgcVCLNavigationBar; Tabs: TsgcVCLNavigationTabs);
begin
  Nav.Theme := utDark; // this bar stays dark, whatever the application theme
  Tabs.AccentColor := sgcUIARGB(255, 220, 38, 38); // a red accent
end;
```

The rail is the folded bar. `Collapsed = True` narrows the bar from `ExpandedWidth` (220 pixels by default) to `CollapsedWidth` (56 by default), an icon rail: the captions are hidden, an item without a `Hint` shows its caption as the hint, and a badge becomes a dot on the icon. `ShowMenuButton = True` adds the usual "hamburger" button at the top that toggles it for the user. With `AnimateCollapse`, `True` by default, the width changes with a short animation at run time; `OnCollapsedChange` tells you when it happened. All sizes are logical pixels, scaled to the monitor, so the bar stays sharp at any DPI.

Main thread only

Like every VCL control, the navigation controls are used from the main thread. The same goes for the theme procedures and for the notification model: the source says so explicitly. When a background thread, a socket or a timer thread wants to update a badge or show a toast, it queues the work to the main thread first, for example with `TThread.Queue` or `TThread.Synchronize`, and the queued code changes the controls. The `TTimer` of demo 05 is fine as it is, because a VCL timer fires on the main thread.

Summary: who does what

Piece	Its job
`TsgcVCLNavigationBar`	Shows the items, keeps the selection, folds groups and the rail, filters with the search box, fires `OnSelectionChange` and `OnItemClick`.
`Nav.Items` (`TsgcNavigationItem`)	The menu: captions, glyphs, badges, groups, tree levels and footer items.
Your code	Shows the page that belongs to the selected item.
`TsgcVCLNavigationTabs`	Shows the `Control` of the active tab, asks your code to add and close tabs.
`TsgcVCLNavigationBreadcrumb`	Shows the path of the selected item of its bar and lets the user jump up.
`TsgcVCLToastManager`	Pops up short messages that close by themselves.
`sgcUISetApplicationTheme` and friends	One look for every sgcUI control and, on request, for the standard controls.

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.NavHello	`NavHello.dpr`	Items typed at design time, `Caption` and `Glyph`, the `OnItemClick` event
02.NavMenuBuilder	`NavMenuBuilder.dpr`	Items added in code, accordion groups, badges, disabled and footer items, search, the menu button, `OnSelectionChange`
03.NavDocTabs	`NavDocTabs.dpr`	The tab strip, the `Control` of a tab, new and close buttons, modified tabs
04.NavAppearance	`NavAppearance.dpr`	The application theme, the style preset, the accent colour, the folded rail
05.NavMailDesk	`NavMailDesk.dpr`	Tree mode, badges from data, the breadcrumb linked to the bar, one tab per document, a toast

> **Note.** The demos need sgcNavigation installed, so the components appear on the SGC Navigation palette page and the `sgcUI\Source` folder is on the library path. They use only features that are part of every edition, and they need no images, database or network: the icons are glyph letters and the data is created in code.

> **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.

4.1 Demo 01: Hello Navigation

Folder `01.NavHello`, project `NavHello.dpr`.

What you will build

The smallest sgcNavigation application. A navigation bar on the left lists five items, Home, Inbox, Calendar, Contacts and Settings, each with a one letter glyph. A large label on the right plays the part of the page and shows the caption of the item you click.

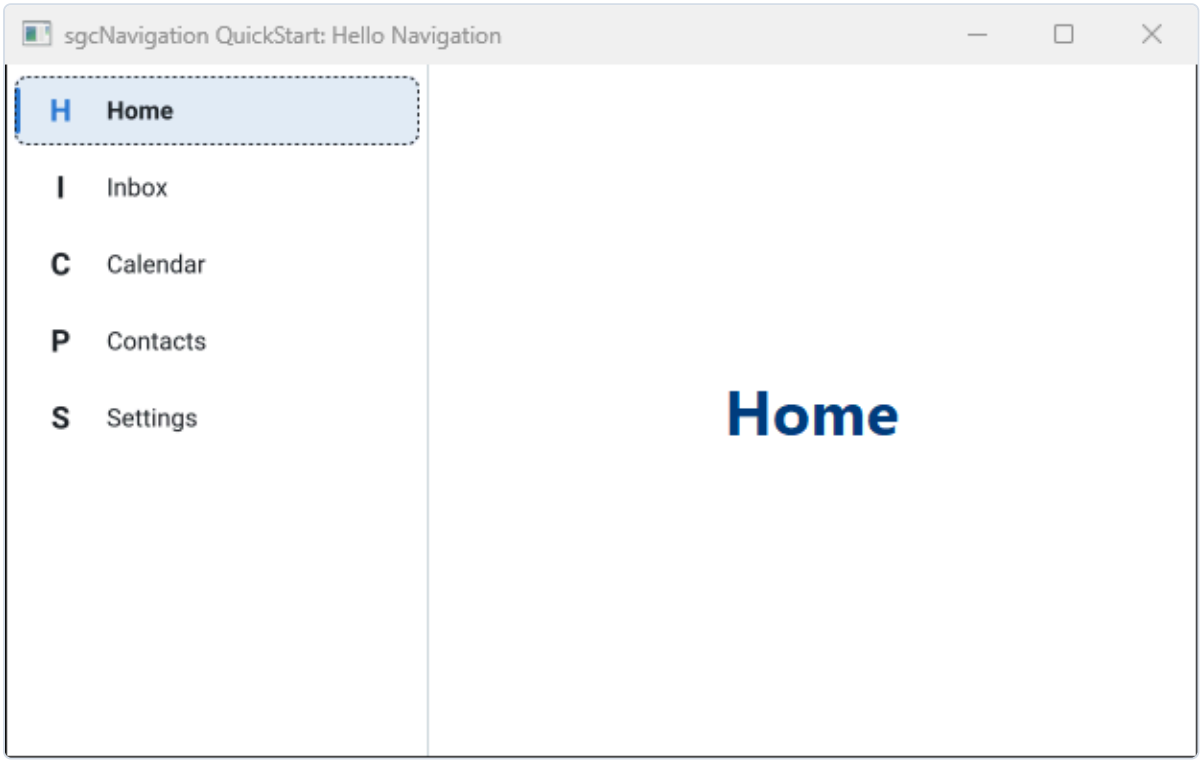


Figure 4.1. Demo 01. The selected item is highlighted with the accent colour; the letters are the glyphs of the items.

Components on the form

Component	Class	Properties you set
`Nav`	`TsgcVCLNavigationBar`	`Items`: five entries with `Caption` and `Glyph`. `ExpandedWidth = 220`, `SelectedIndex = 0`, `OnItemClick = NavItemClick`. `Align` is already `alLeft` by default.
`lblPage`	`TLabel`	`Align = alClient`, centred, bold 24 point text.

The bar has no `Align` line in the form file because `alLeft` is its default: dropping a bar on a form already docks it to the left edge. The rest of its DFM, the five items, is shown in chapter 1.

The code

The `uses` clause names the unit of the items and the unit of the bar:

```
uses
  Classes, Controls, Forms, StdCtrls,
  // sgc
  sgcNavigation_Model, sgcNavigation_VCL;
```

And this is the only event handler of the demo:

```
procedure TfrmNavHello.NavItemClick(Sender: TObject;
  aItem: TsgcNavigationItem);
begin
  { aItem is the TsgcNavigationItem that was clicked: one entry of
    Nav.Items, with the Caption set in the Object Inspector. A real
    application would show the page of that item here. }
  lblPage.Caption := aItem.Caption;
end;
```

How it works

1. The form loads from its DFM. The bar reads its five items and selects item 0, Home, because of `SelectedIndex = 0`. The label starts with the caption Home, also set in the DFM.
2. When you click Inbox, the bar sets `SelectedIndex` to 1, highlights the row and fires `OnItemClick` with the Inbox item.
3. `NavItemClick` copies `aItem.Caption` into the label.
4. The keyboard works too: click the bar, move with the arrow keys and press Enter on an item. Enter fires the same event as a click.

Try this

- Set `Collapsed = True` on `Nav`. The bar folds into a narrow rail of glyphs; hover over a glyph to see the caption as a hint.
- Set the `Badge` of the Inbox item to `5` in the collection editor. A small counter appears on the right of the row.
- Set `Nav.Theme` to `utDark`.

4.2 Demo 02: Menu Builder

Folder `02.NavMenuBuilder`, project `NavMenuBuilder.dpr`.

What you will build

In demo 01 the items were typed in the Object Inspector. Most applications build the menu in code instead, from their modules, the user rights or a configuration file. This demo builds a menu with three groups in `FormCreate`: Sales, Products and Reports. It uses badges, a disabled item, a footer item, a search box and the menu button that folds the bar, and the page on the right shows the caption and the hint of the selected item.

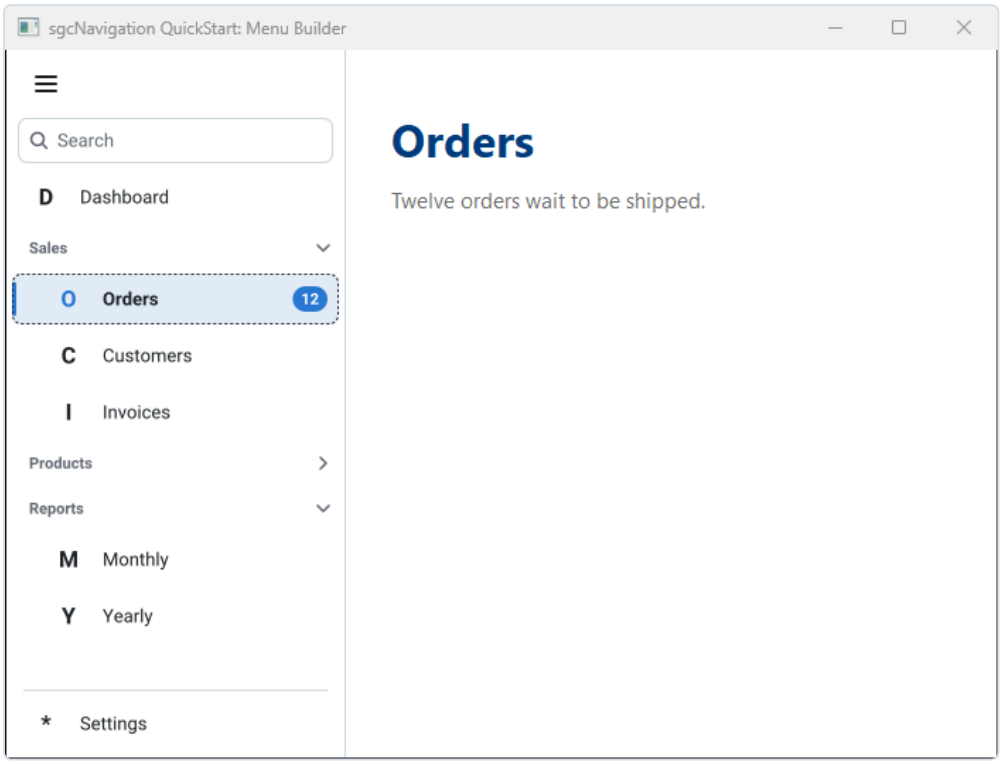


Figure 4.2. Demo 02. Orders is selected and carries the badge 12. The Products group starts folded, and Settings is pinned in the footer.

Components on the form

Component	Class	Properties you set
`Nav`	`TsgcVCLNavigationBar`	`Mode = nbmAccordion`, `AllowMultiExpand = True`, `ShowSearch = True`, `ShowMenuButton = True`, `ExpandedWidth = 240`, `Items` empty, `OnSelectionChange = NavSelectionChange`.
`pnlPage`	`TPanel`	`Align = alClient`, no bevel. Holds the two labels.
`lblTitle`	`TLabel`	The caption of the selected item, bold.
`lblInfo`	`TLabel`	The hint of the selected item, word wrapped.

```
  object Nav: TsgcVCLNavigationBar
    Left = 0
    Top = 0
    Width = 240
    Height = 500
    Items = <>
    Mode = nbmAccordion
    ExpandedWidth = 240
    AllowMultiExpand = True
    ShowSearch = True
    ShowMenuButton = True
    TabOrder = 0
    OnSelectionChange = NavSelectionChange
  end
```

The code

A small helper adds an item with a caption, a glyph and a hint, and returns it so the caller can set more properties on the result:

```
function TfrmNavMenuBuilder.DoAdd(const aCaption, aGlyph, aHint: string)
  : TsgcNavigationItem;
begin
  { AddItem returns the new item: set its other properties on the result }
  Result := Nav.Items.AddItem(aCaption, aGlyph);
  Result.Hint := aHint;
end;
```

`FormCreate` builds the whole menu inside one update bracket, then selects Orders:

```
procedure TfrmNavMenuBuilder.FormCreate(Sender: TObject);
begin
  { BeginUpdate and EndUpdate bracket the changes, so the bar is laid out
    and painted once, at EndUpdate, and not after every item. }
  Nav.Items.BeginUpdate;
  try
    Nav.Items.Clear;
    DoAdd('Dashboard', 'D', 'The figures of the day at a glance.');

    { a group header: the third parameter of AddItem is IsGroup }
    Nav.Items.AddItem('Sales', '', True);
    DoAdd('Orders', 'O', 'Twelve orders wait to be shipped.').Badge := '12';
    DoAdd('Customers', 'C', 'Everyone who bought from us.');
    DoAdd('Invoices', 'I', 'Issued, paid and overdue invoices.');

    { a group can start folded: click its header to unfold it }
    Nav.Items.AddItem('Products', '', True).Expanded := False;
    DoAdd('Catalog', 'K', 'Every product we sell.');
    DoAdd('Stock', 'S', 'Three products are running low.').Badge := '3';
    { a disabled item is painted greyed and can not be selected }
    DoAdd('Suppliers', 'U', 'Not licensed in this demo.').Enabled := False;

    Nav.Items.AddItem('Reports', '', True);
    DoAdd('Monthly', 'M', 'Sales of the current month.');
    DoAdd('Yearly', 'Y', 'Sales of the current year.');

    { nipFooter pins the item at the bottom of the bar }
    DoAdd('Settings', '*', 'Preferences of the application.').Placement :=
      nipFooter;
```

```
  finally
    Nav.Items.EndUpdate;
  end;

  { select Orders (index 2: Dashboard is 0, the Sales header is 1) }
  Nav.SelectedIndex := 2;
end;
```

And the page follows the selection:

```
procedure TfrmNavMenuBuilder.NavSelectionChange(Sender: TObject);
var
  oItem: TsgcNavigationItem;
begin
  if Nav.SelectedIndex < 0 then
    Exit;
  oItem := Nav.Items[Nav.SelectedIndex];
  lblTitle.Caption := oItem.Caption;
  lblInfo.Caption := oItem.Hint;
end;
```

How it works

1. The collection is flat. `AddItem('Sales', '', True)` adds a group header, and the items added after it belong to Sales until the next header, Products.
2. Calls such as `DoAdd(...).Badge := '12'` set one more property on the item just added. The same pattern sets `Expanded := False` on the Products header, `Enabled := False` on Suppliers and `Placement := nipFooter` on Settings.
3. `EndUpdate` lays the bar out once. Then `Nav.SelectedIndex := 2` selects Orders: Dashboard is item 0 and the Sales header is item 1, because group headers count in the index.
4. Every change of the selection, by a click, the keyboard or code, fires `OnSelectionChange`, and `NavSelectionChange` reads the selected item from `Nav.Items[Nav.SelectedIndex]`.
5. The search box filters the items as you type, and the menu button at the top folds the bar into the rail and back.

Try this

- Set `Mode = nbmSidebar`. The group headers become static labels, and Products can no longer be folded.
- Set `FilterMode = nfmFuzzy`, run, and type `ivc` in the search box: Invoices is found.
- Set `AllowMultiExpand = False`. Opening one group now closes the others.

4.3 Demo 03: Document Tabs

Folder `03.NavDocTabs`, project `NavDocTabs.dpr`.

What you will build

The second navigation component of sgcNavigation: a tab strip for documents. Three tabs, Welcome, Notes and To do, each show their own memo. The + button opens a new document, the close button removes a tab (the last one stays open), and typing in a memo marks its tab as modified. A status line at the bottom reports what happened.

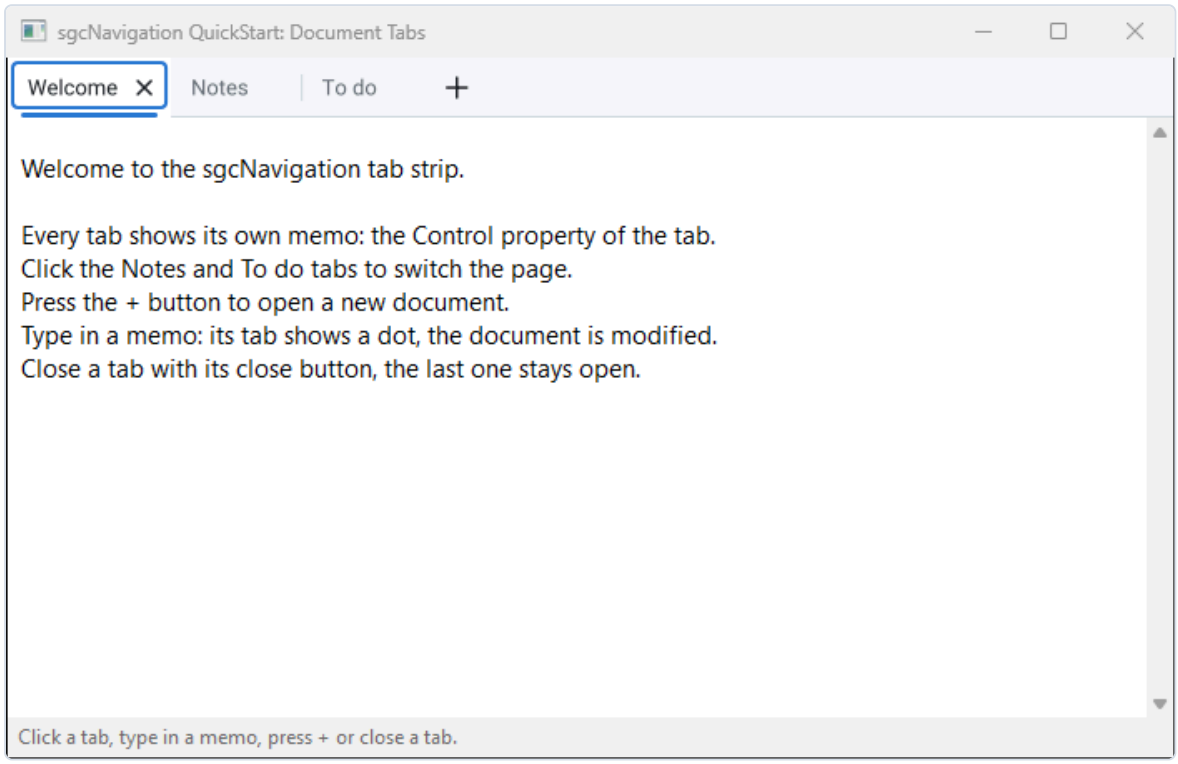


Figure 4.3. Demo 03. Each tab shows its own memo; the + button after the last tab adds a document.

Components on the form

Component	Class	Properties you set
`Tabs`	`TsgcVCLNavigationTabs`	`Tabs`: Welcome, Notes and To do, with `Control` = `memoWelcome`, `memoNotes` and `memoTodo`. `ActiveIndex = 0`, `CloseButtons = tcbHover`, events `OnTabChange`, `OnNewTab`, `OnTabClose`. `Align` is `alTop` and `ShowNewButton` is `True` by default.
`memoWelcome`, `memoNotes`, `memoTodo`	`TMemo`	`Align = alClient`, no border, all three with `OnChange = MemoChange`.
`lblStatus`	`TLabel`	`Align = alBottom`, written by the events.

```
  object Tabs: TsgcVCLNavigationTabs
    Tabs = <
      item
        Caption = 'Welcome'
        Control = memoWelcome
      end
      item
        Caption = 'Notes'
        Control = memoNotes
      end
      item
        Caption = 'To do'
        Control = memoTodo
      end>
    ActiveIndex = 0
    CloseButtons = tcbHover
    OnTabChange = TabsTabChange
    OnTabClose = TabsTabClose
    OnNewTab = TabsNewTab
    // ...
  end
```

The code

The `uses` clause names the unit of the tab class and the unit of the VCL strip:

```
uses
  SysUtils, Classes, Controls, Forms, StdCtrls,
  // sgc
  sgcNavigation_Tabs, sgcNavigation_Tabs_VCL;
```

The status line follows the active tab:

```
procedure TfrmNavDocTabs.TabsTabChange(Sender: TObject);
begin
  if Tabs.ActiveIndex >= 0 then
    lblStatus.Caption := 'Active tab: ' + Tabs.Tabs[Tabs.ActiveIndex].Caption
      + '   (' + IntToStr(Tabs.Tabs.Count) + ' open)';
end;
```

The + button creates a memo, adds a tab for it and activates the tab:

```
procedure TfrmNavDocTabs.TabsNewTab(Sender: TObject);
var
  oMemo: TMemo;
  oTab: TsgcNavigationTab;
begin
  { the page of the new tab: a memo created in code, like the three of the
    DFM }
  oMemo := TMemo.Create(Self);
  oMemo.Parent := Self;
  oMemo.Align := alClient;
  oMemo.BorderStyle := bsNone;
  oMemo.ScrollBars := ssVertical;
  oMemo.OnChange := MemoChange;

  Inc(FCount);
  oTab := Tabs.Tabs.AddTab('Untitled ' + IntToStr(FCount));
  oTab.Control := oMemo;
  { activating the tab shows its memo and fires OnTabChange }
  Tabs.ActiveIndex := oTab.Index;
  oMemo.SetFocus;
end;
```

A close button asks first, then frees the memo of the tab:

```
procedure TfrmNavDocTabs.TabsTabClose(Sender: TObject; aIndex: Integer;
  var aAllow: Boolean);
var
  oControl: TComponent;
begin
  { keep at least one document open }
  aAllow := Tabs.Tabs.Count > 1;
  if not aAllow then
  begin
    lblStatus.Caption := 'The last tab stays open.';
    Exit;
  end;
  { the strip removes the tab, the memo of the tab is freed here }
  oControl := Tabs.Tabs[aIndex].Control;
  Tabs.Tabs[aIndex].Control := nil;
  oControl.Free;
end;
```

And typing marks the tab of the memo as modified:

```
procedure TfrmNavDocTabs.MemoChange(Sender: TObject);
var
  i: Integer;
begin
  { the memo that changed is the Control of one tab: mark that tab }
  for i := 0 to Tabs.Tabs.Count - 1 do
    if Tabs.Tabs[i].Control = Sender then
      Tabs.Tabs[i].Modified := True;
end;
```

How it works

1. The three memos are all aligned `alClient` and lie on top of each other. The strip shows the `Control` of the active tab and hides the other two, which is why the demo has no code to switch pages.

2. A click on the + button fires `OnNewTab`. The strip adds nothing by itself; `TabsNewTab` creates the memo, calls `AddTab` and sets `ActiveIndex`, which shows the new memo and fires `OnTabChange`.
3. A click on a close button fires `OnTabClose` with the index of the tab. With one tab left, `aAllow := False` keeps it. Otherwise the handler takes the memo away from the tab, frees it, and the strip removes the tab when the event returns.
4. `MemoChange` finds the tab whose `Control` is the memo that changed and sets `Modified := True`, which draws a dot in its close slot.

> **Tip.** Set `Control := nil` before you free the control, as the demo does. The tab then no longer points to a control that is about to disappear.

Try this

- Set `TabPosition = ntpLeft`. The strip moves to the left edge; set its `Align` to `alLeft` too.
- Set `ShowListButton = True`. A button at the end of the strip opens a menu with every tab.
- Set `TabSizing = tsScroll` and open many tabs with +. The tabs keep their width and the strip scrolls.

4.4 Demo 04: Appearance

Folder `04.NavAppearance`, project `NavAppearance.dpr`.

What you will build

A form with a navigation bar, a tab strip and a few standard VCL controls, and a toolbar of three combo boxes that change the look of all of them at once: the theme (Light, Dark, High contrast), the style (Classic, Fluent) and the accent colour (Automatic, Blue, Green, Purple, Orange). A check box folds the bar into the icon rail. The demo starts in Dark, Fluent and Purple.

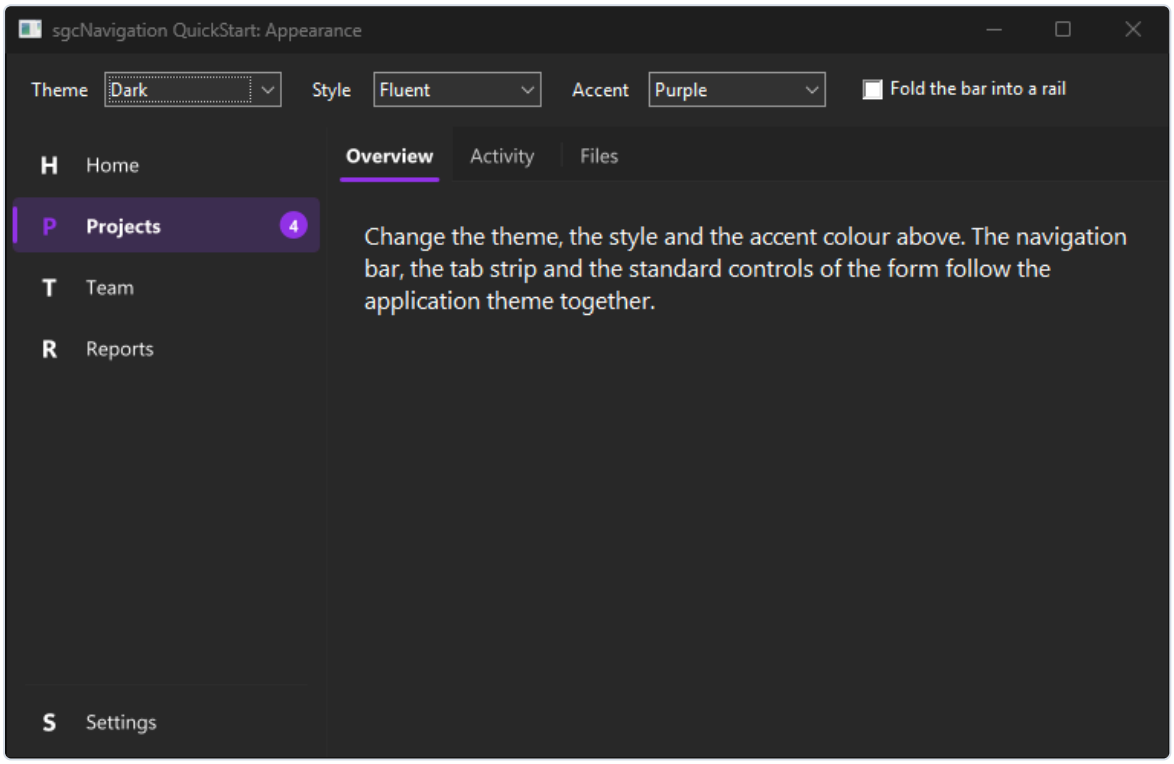


Figure 4.4. Demo 04 in its starting look. The bar, the tabs, the combo boxes and the panels all follow the same dark theme and the purple accent.

Components on the form

Component	Class	Properties you set
`pnlTop`	`TPanel`	Holds `cbTheme` (`ItemIndex = 1`), `cbStyle` (1) and `cbAccent` (3), all with `OnChange = AppearanceChange`, and the check box `chkRail` (`OnClick = chkRailClick`).
`Nav`	`TsgcVCLNavigationBar`	Items Home, Projects (`Badge = '4'`), Team, Reports and Settings (`Placement = nipFooter`). `ExpandedWidth = 210`, `SelectedIndex = 1`.
`Tabs`	`TsgcVCLNavigationTabs`	Tabs Overview, Activity and Files, each with `Closable = False`. `ShowNewButton = False`, `ActiveTabFont.Bold = True`.
`pnlBody`, `lblContent`	`TPanel`, `TLabel`	The page, with a word wrapped text.

Because this strip only shows views and never documents, every tab has `Closable = False` and the + button is switched off:

```
  object Tabs: TsgcVCLNavigationTabs
    Tabs = <
      item
        Caption = 'Overview'
        Closable = False
      end
      // ...
      item
        Caption = 'Files'
        Closable = False
      end>
    ActiveIndex = 0
    ShowNewButton = False
    ActiveTabFont.Bold = True
  end
```

The code

The `uses` clause adds the two theme units of sgcUICore:

```
uses
  Classes, Controls, Forms, StdCtrls, ExtCtrls,
  // sgc
  sgcUICore_Theme, sgcUICore_Theme_VCL, sgcNavigation_Model,
  sgcNavigation_VCL, sgcNavigation_Tabs, sgcNavigation_Tabs_VCL;
```

`FormCreate` applies the starting choices of the combo boxes, and every combo box calls the same handler:

```
procedure TfrmNavAppearance.FormCreate(Sender: TObject);
begin
  { apply the choices of the combo boxes, set in the DFM }
  AppearanceChange(nil);
end;

procedure TfrmNavAppearance.AppearanceChange(Sender: TObject);
begin
  { 1. the style, before the theme so the theme is drawn with it }
  if cbStyle.ItemIndex = 1 then
    sgcUISetApplicationThemePreset(CS_UI_THEME_FLUENT)
  else
    sgcUISetApplicationThemePreset(CS_UI_THEME_CLASSIC);

  { 2. the accent colour, $AARRGGBB: sgcUIARGB(Alpha, Red, Green, Blue) }
  case cbAccent.ItemIndex of
    1:
      sgcUISetApplicationAccentColor(sgcUIARGB(255, 37, 99, 235)); // blue
    2:
      sgcUISetApplicationAccentColor(sgcUIARGB(255, 22, 163, 74)); // green
    3:
      sgcUISetApplicationAccentColor(sgcUIARGB(255, 147, 51, 234)); // purple
    4:
      sgcUISetApplicationAccentColor(sgcUIARGB(255, 234, 88, 12)); // orange
  else
    sgcUISetApplicationAccentColor(0); // automatic
  end;

  { 3. the theme of every sgcUI control and, with True, of the standard
    controls of the form }
  case cbTheme.ItemIndex of
    1:
```

```
      sgcUISetApplicationTheme(utDark, True);
    2:
      sgcUISetApplicationTheme(utHighContrast, True);
  else
    sgcUISetApplicationTheme(utLight, True);
  end;
end;
```

The check box folds the bar:

```
procedure TfrmNavAppearance.chkRailClick(Sender: TObject);
begin
  Nav.Collapsed := chkRail.Checked;
end;
```

How it works

1. `Nav` and `Tabs` keep their `Theme` at the default, `utDefault`, so both follow the application theme.
2. `AppearanceChange` first sets the style preset, then the accent colour, then the theme, so the theme is drawn in the chosen style with the chosen accent.
3. `sgcUISetApplicationTheme(utDark, True)` repaints every sgcUI control that follows the application theme. The `True` also themes the standard controls of every open form, such as `pnlTop`, the combo boxes and the labels; this part comes from `sgcUICore_Theme_VCL`, which is why that unit is in the `uses` clause.
4. The accent colour is an ARGB value: `sgcUIARGB(255, 147, 51, 234)` is an opaque purple. It colours the selection bar, the active tab underline and the badges. 0 returns to the automatic accent.
5. `chkRailClick` sets `Nav.Collapsed`, and the bar animates to its narrow width because `AnimateCollapse` is `True` by default.

Try this

- Set `Nav.Theme = utLight` in the Object Inspector and run. Switch the combo box to Dark: the bar stays light, because a control with its own theme no longer follows the application.
- Add `Tabs.AccentColor := sgcUIARGB(255, 220, 38, 38);` at the end of `FormCreate`. The tabs use red, the bar keeps the application accent.
- Set `Nav.AnimateCollapse = False`. The rail check box now folds the bar at once.

4.5 Demo 05: Mail Desk

Folder `05.NavMailDesk`, project `NavMailDesk.dpr`.

What you will build

A small mail client that puts the previous demos together. The bar is a tree of folders, with Customers and Suppliers under Inbox, and every folder shows its number of unread messages as a badge. A breadcrumb above the list shows the selected folder and its parents. The Messages tab holds a list of the messages of the folder, unread ones in bold; a double click opens a message in a tab of its own. After 1.5 seconds a new message arrives and a toast announces it.

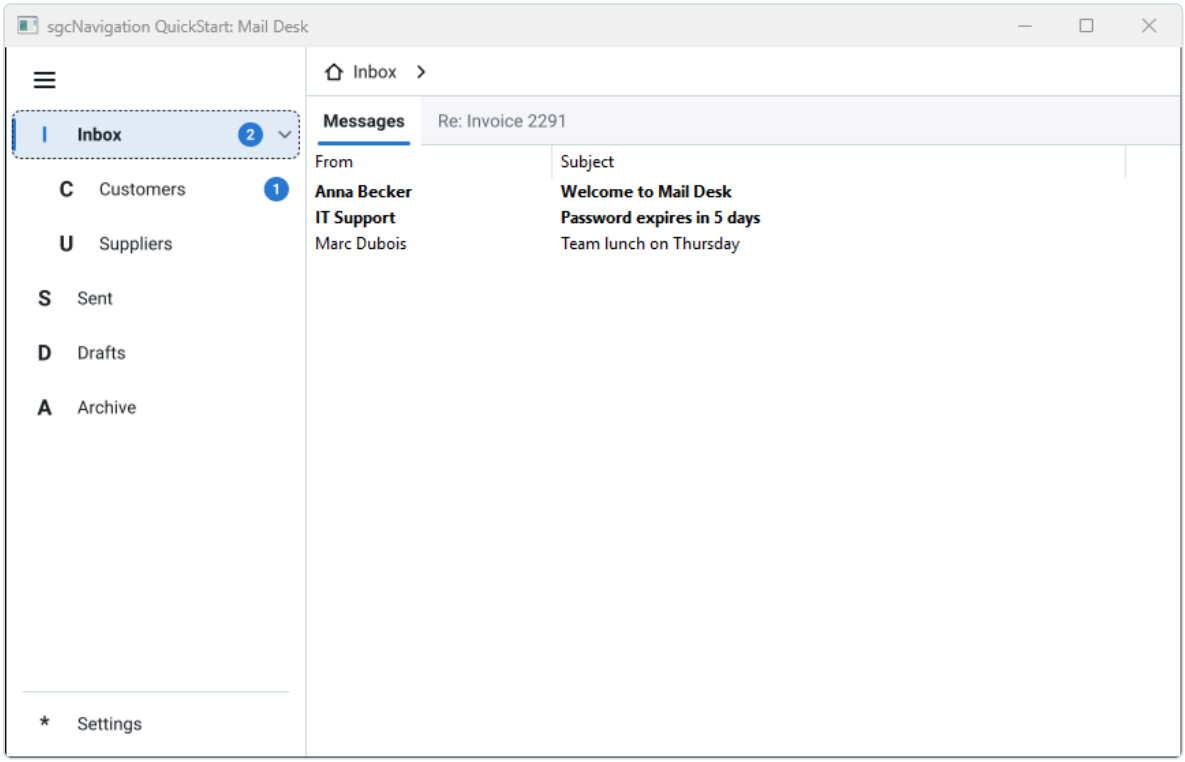


Figure 4.5. Demo 05 right after the start. One message is already open in its own tab. The toast is a separate window and is not part of this window capture.

Components on the form

Component	Class	Properties you set
`Nav`	`TsgcVCLNavigationBar`	`Mode = nbmTree`, `ShowMenuButton = True`, `SelectedIndex = 0`. Items Inbox, Customers and Suppliers (`Depth = 1`), Sent, Drafts, Archive, Settings (`Placement = nipFooter`). `OnSelectionChange = NavSelectionChange`.
`pnlMain`	`TPanel`	`Align = alClient`. Holds the breadcrumb, the tabs, the list and the message memos.
`Crumb`	`TsgcVCLNavigationBreadcrumb`	`NavigationBar = Nav`. Nothing else.
`Tabs`	`TsgcVCLNavigationTabs`	One tab, Messages, with `Closable = False` and `Control = lvMessages`. `ShowNewButton = False`, `CloseButtons = tcbHover`, `ActiveTabFont.Bold = True`, `OnTabClose = TabsTabClose`.
`lvMessages`	`TListView`	`ViewStyle = vsReport` with From and Subject columns, `ReadOnly`, `RowSelect`, `OnDblClick`, `OnCustomDrawItem`.
`Toasts`	`TsgcVCLToastManager`	Defaults: bottom right corner, 5 seconds.
`tmrNewMail`	`TTimer`	`Interval = 1500`, `OnTimer = tmrNewMailTimer`.

```
object Nav: TsgcVCLNavigationBar
  Items = <
    item
      Caption = 'Inbox'
      Glyph = 'I'
    end
    item
      Caption = 'Customers'
      Glyph = 'C'
      Depth = 1
    end
    item
      Caption = 'Suppliers'
      Glyph = 'U'
      Depth = 1
    end
    // ...
    item
      Caption = 'Settings'
      Glyph = '*'
      Placement = nipFooter
    end>
  Mode = nbmTree
  // ...
end
// ...
  object Crumb: TsgcVCLNavigationBreadcrumb
    NavigationBar = Nav
  end
```

The code

The messages are kept in a plain array of records. In a real application they would come from a mail server or a database:

```
type
  TMailMessage = record
    Folder: Integer; // the index of the folder in Nav.Items
    From: string;
    Subject: string;
    Body: string;
    Unread: Boolean;
  end;

  // ...
  private
    FMails: array of TMailMessage;
    { FRows[i] is the index in FMails of row i of lvMessages }
    FRows: array of Integer;
```

`FormCreate` adds the messages, sets the badges, fills the list and opens one message:

```
procedure TfrmNavMailDesk.FormCreate(Sender: TObject);
begin
  DoAddMail(CS_INBOX, 'Anna Becker', 'Welcome to Mail Desk',
    'Hi,'#13#10#13#10'this is a small mail client built with sgcNavigation.' +
    #13#10'Double click a message to open it in its own tab.', True);
  // ...
  DoAddMail(CS_ARCHIVE, 'Anna Becker', 'Project kickoff',
    'The project starts next week.', False);
  DoUpdateBadges;
  { SelectedIndex = 0 (Inbox) is set in the DFM: show its messages }
  DoFillList;
  { open one message so the tab strip shows a second tab, but keep the
    message list in front }
  DoOpenMail(3);
  Tabs.ActiveIndex := 0;
end;
```

The badge of every folder is its number of unread messages:

```
procedure TfrmNavMailDesk.DoUpdateBadges;
var
  i, j, vCount: Integer;
begin
  { the Badge of a folder is its number of unread messages, empty hides it }
  for i := 0 to Nav.Items.Count - 1 do
  begin
    vCount := 0;
    for j := 0 to Length(FMails) - 1 do
      if (FMails[j].Folder = i) and FMails[j].Unread then
        Inc(vCount);
    if vCount > 0 then
      Nav.Items[i].Badge := IntToStr(vCount)
    else
      Nav.Items[i].Badge := '';
  end;
end;
```

The list shows the messages of the selected folder, and a change of folder refills it:

```
procedure TfrmNavMailDesk.DoFillList;
var
  i: Integer;
  oItem: TListItem;
begin
  lvMessages.Items.BeginUpdate;
  try
    lvMessages.Items.Clear;
    SetLength(FRows, 0);
    for i := 0 to Length(FMails) - 1 do
      if FMails[i].Folder = Nav.SelectedIndex then
      begin
        oItem := lvMessages.Items.Add;
        oItem.Caption := FMails[i].From;
        oItem.SubItems.Add(FMails[i].Subject);
        SetLength(FRows, Length(FRows) + 1);
        FRows[Length(FRows) - 1] := i;
      end;
  finally
    lvMessages.Items.EndUpdate;
  end;
end;

procedure TfrmNavMailDesk.NavSelectionChange(Sender: TObject);
begin
  { another folder: show its messages on the Messages tab. The breadcrumb
    follows the bar by itself. }
  DoFillList;
  Tabs.ActiveIndex := 0;
end;
```

Opening a message reuses its tab when it is already open, otherwise it creates a read only memo and a tab for it. The `Tag` of the tab remembers which message it shows:

```
procedure TfrmNavMailDesk.DoOpenMail(aMail: Integer);
var
  i: Integer;
  oMemo: TMemo;
  oTab: TsgcNavigationTab;
begin
  { the message may already have a tab: the Tag of the tab is the index of
    the message in FMails }
  for i := 1 to Tabs.Tabs.Count - 1 do
    if Tabs.Tabs[i].Tag = aMail then
    begin
      Tabs.ActiveIndex := i;
      Exit;
    end;

  oMemo := TMemo.Create(Self);
  oMemo.Parent := pnlMain;
  oMemo.Align := alClient;
  oMemo.BorderStyle := bsNone;
  oMemo.ReadOnly := True;
  oMemo.ScrollBars := ssVertical;
  oMemo.Font.Height := -15;
  oMemo.Lines.Text := 'From: ' + FMails[aMail].From + #13#10 + 'Subject: ' +
    FMails[aMail].Subject + #13#10#13#10 + FMails[aMail].Body;

  oTab := Tabs.Tabs.AddTab(FMails[aMail].Subject);
```

```
  oTab.Tag := aMail;
  oTab.Control := oMemo;
  Tabs.ActiveIndex := oTab.Index;

  { an opened message is read: refresh the counters and the bold rows }
  FMails[aMail].Unread := False;
  DoUpdateBadges;
  lvMessages.Invalidate;
end;
```

A double click on a row opens its message, unread rows are drawn in bold, and closing a message tab frees its memo:

```
procedure TfrmNavMailDesk.lvMessagesDblClick(Sender: TObject);
begin
  if lvMessages.Selected <> nil then
    DoOpenMail(FRows[lvMessages.Selected.Index]);
end;

procedure TfrmNavMailDesk.lvMessagesCustomDrawItem(Sender: TCustomListView;
  Item: TListItem; State: TCustomDrawState; var DefaultDraw: Boolean);
begin
  { unread messages in bold }
  if FMails[FRows[Item.Index]].Unread then
    Sender.Canvas.Font.Style := [fsBold]
  else
    Sender.Canvas.Font.Style := [];
end;

procedure TfrmNavMailDesk.TabsTabClose(Sender: TObject; aIndex: Integer;
  var aAllow: Boolean);
var
  oControl: TComponent;
begin
  { Messages has Closable = False, so only a message tab gets here: free
    its memo, the strip removes the tab }
  oControl := Tabs.Tabs[aIndex].Control;
  Tabs.Tabs[aIndex].Control := nil;
  oControl.Free;
end;
```

Finally the timer delivers a new message and shows a toast:

```
procedure TfrmNavMailDesk.tmrNewMailTimer(Sender: TObject);
begin
  tmrNewMail.Enabled := False;
  DoAddMail(CS_CUSTOMERS, 'Kenji Sato', 'Order confirmed',
    'We confirm the order of 25 licences.', True);
  DoUpdateBadges;
  if Nav.SelectedIndex = CS_CUSTOMERS then
    DoFillList;
  { a toast in the corner of the form, closed after Duration milliseconds }
  Toasts.Show('New message', 'Kenji Sato: Order confirmed', nsvInfo);
end;
```

How it works

1. In tree mode, Customers and Suppliers have `Depth = 1` and follow Inbox, so they are its children and fold with it. The flat index of each folder is also its folder number in the records (`CS_INBOX = 0` , `CS_CUSTOMERS = 1` and so on), which keeps the code simple.
2. The breadcrumb has `NavigationBar = Nav` and no code at all. When you select Customers it shows *Inbox* > *Customers*, and a click on the Inbox segment selects Inbox in the bar, which fires `OnSelectionChange` and refills the list.
3. The Messages tab has `Closable = False` , so it never shows a close button and `TabsTabClose` only ever sees message tabs.
4. `DoOpenMail` uses the `Tag` of each tab to find a message that is already open. A new message tab gets a read only memo as its `Control` , and the strip shows that memo while the tab is active, exactly as in demo 03.
5. Reading a message clears its unread flag; `DoUpdateBadges` recomputes the counters (an empty `Badge` hides it) and `lvMessages.Invalidate` redraws the rows without bold.
6. The timer fires once on the main thread, adds a message to Customers, updates the badges and calls `Toasts.Show` . The toast appears in the bottom right corner of the form and closes after five seconds.

Try this

- Set `Toasts.Position = tpsTopRight` and `Toasts.Duration = 10000` in the Object Inspector. The toast moves to the top right corner and stays ten seconds.
- Change `nsvInfo` to `nsvSuccess` or `nsvWarning` in `tmrNewMailTimer` .
- Set `Nav.ShowSearch = True` to search the folders, or add `sgcUISetApplicationTheme(utDark, True);` to `FormCreate` (with `sgcUICore_Theme` and `sgcUICore_Theme_VCL` in the `uses` clause) for a dark mail client.

5. Main components

The SGC Navigation palette page holds a handful of controls, and four of them cover a typical first application: the bar, the tabs, the breadcrumb and the toast manager. This chapter introduces each of them, then the components you will want later.

Every class has its own help topic, named after the class, such as `TsgcVCLNavigationBar.htm` or `TsgcNavigationItem.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
`TsgcVCLNavigationBar`	SGC Navigation	The side menu	`Items`, `Mode`, `SelectedIndex`, `Collapsed`, `ShowSearch`, `ShowMenuButton`, `OnSelectionChange`, `OnItemClick`
`TsgcNavigationItem`	none, an item of `Items`	One row of the bar	`Caption`, `Glyph`, `ImageIndex`, `Badge`, `IsGroup`, `Depth`, `Expanded`, `Placement`, `Tag`
`TsgcVCLNavigationTabs`	SGC Navigation	A tab strip for documents or views	`Tabs`, `ActiveIndex`, `TabPosition`, `CloseButtons`, `ShowNewButton`, `OnTabChange`, `OnTabClose`, `OnNewTab`
`TsgcNavigationTab`	none, an item of `Tabs`	One tab	`Caption`, `Control`, `Closable`, `Modified`, `Tag`
`TsgcVCLNavigationNotebook`	SGC Navigation	A page control on the tab strip, pages designed in the form	`ActivePage`, `TabPosition`, `MultiLine`, `OnPageChanging`, `OnPageChange`
`TsgcVCLNavigationBreadcrumb`	SGC Navigation	The path of the current location	`NavigationBar`, `Segments`, `Path`, `OnSegmentClick`
`TsgcVCLToastManager`	SGC Navigation	Short pop-up notifications	`Show`, `Position`, `Duration`, `Anchor`, `MaxVisible`, `Model`
`TsgcNotificationModel`	SGC Navigation	The list of notifications with their read state	`Add`, `Post`, `OnNotification`, `OnUnreadCountChange`
`TsgcVCLNotificationInbox`	SGC Navigation	A visible list of the notifications of a model	`Model`
`TsgcVCLCommandPalette`	SGC Navigation	Run commands by typing a few letters (not in Community)	See its help topic
`TsgcVCLDockManager`	SGC Navigation	IDE style docking workspace (not in Community)	See its help topic

Class	Palette page	What it is for	Key members
`TsgcFMXNavigationBar` and the other `TsgcFMX...` classes	SGC Navigation (FireMonkey)	The FireMonkey twins	The same as the VCL classes

The navigation bar

TsgcVCLNavigationBar

The control you drop first. It is aligned to the left by default and paints one row per shown item.

- **Content:** `Items`, `Images` for real icons (with `ImageIndex` on the items), `SelectedIndex`.
- **Layout:** `Mode` (`nbmSidebar`, `nbmAccordion`, `nbmTree`), `AllowMultiExpand`, `ExpandedWidth`, `CollapsedWidth`, `ItemHeight`, `GroupHeight`, `Indent`.
- **Rail:** `Collapsed`, `ShowMenuButton`, `AnimateCollapse`, `OnCollapsedChange`.
- **Search:** `ShowSearch`, `SearchPlaceholder`, `Filter`, `FilterMode`.
- **Look:** `Theme`, `AccentColor`.
- **Events:** `OnSelectionChange`, `OnItemClick`, and the `OnClick` of each item.
- **Methods:** `ItemAtPos`, `ItemRect` and `ScrollIntoView`; the `Model` property gives access to the rest, such as `Model.ExpandAll`, `Model.CollapseAll` and `Model.ItemPath`.

The keyboard works without any code: the arrow keys move the focus, Enter activates, Left and Right fold and unfold a group or a tree parent, and typing jumps to the next item whose caption starts with the typed text (or edits the filter when the search box is shown).

Tabs and the notebook

TsgcVCLNavigationTabs

A tab strip, aligned to the top by default. The tabs live in `Tabs`, and each one can carry a `Control` that the strip shows while the tab is active. `TabPosition` puts the strip on any edge, `TabSizing` (`tsFit`, `tsShrink`, `tsScroll`) decides what happens when the tabs do not fit, `MultiLine` wraps them into rows, and `ShowListButton` adds a menu with every tab. `HandleCloseShortcuts` lets Ctrl+W and Ctrl+F4 close the active tab, and Ctrl+Tab switches tabs.

TsgcVCLNavigationNotebook

When the pages are known at design time, the notebook is the easier choice. It works like a `TPageControl`: right click it, choose **New Page**, and drop controls on the page. Each page has a `Caption`, an `ImageIndex`, a `TabColor`, a `TabHint` and can be hidden or disabled. `OnPageChanging` can refuse to leave a page, which is handy for a form that must be filled in first. The `Notebook` demo of chapter 7 shows it.

The breadcrumb

TsgcVCLNavigationBreadcrumb

Aligned to the top by default. With `NavigationBar` set it follows the bar, as in demo 05. Without it, you describe the path yourself through `Segments` or `Path` (with `PathDelimiter`) and react in `OnSegmentClick`; `OnGetChildren` fills the drop down list of a segment. `AutoTruncate` shortens a long path to fit, and `ShowRootIcon` draws an icon on the first segment, its image or a home glyph.

Notifications

TsgcVCLToastManager

A non visual component. `Show(Title, Body, Severity, Duration, ImageIndex)` pops up a standalone toast, as in demo 05. The look and the behaviour are properties: `Position`, `Anchor`, `Duration`, `MaxVisible`, `PauseOnHover`, `ShowClose`, `Width`, `Margin`, `Spacing`, `Theme`. `OnToastClick` and `OnToastClose` tell you what the user did.

TsgcNotificationModel, TsgcVCLNotificationInbox, TsgcNotificationPreferences

The notification centre. The model keeps the notifications, the inbox shows them with their read state, and a toast manager with its `Model` set pops up the new ones (`AutoShow`). The preferences, edited by the user in a `TsgcVCLNotificationPrefsEditor`, decide per group which channels are on. Add notifications from the main thread only.

Beyond the basics

- **`TsgcVCLCommandPalette`** opens with a keyboard shortcut and runs any command by typing a few letters, over the items of a bar, the actions of a `TActionList` (through `TsgcVCLActionListCommandSource`) and your own commands. Registered and Trial editions; see the `NavigationShell` demo.
- **`TsgcVCLDockManager`** arranges tool panels and documents in an IDE style workspace: docked, tabbed, auto hidden and floating, with the layout saved as JSON. Registered and Trial editions; see the `DockingShell` demo.
- **Action links.** In the Registered and Trial editions the `Action` property of an item links it to a `TAction`, so the caption, the enabled state and the click follow the action. See the `Accordion` demo.
- **Headless images.** `sgcNavigationSaveToPNG` and `sgcNavigationSaveToSVG` in `sgcNavigation_Render` write a navigation model to an image with no window.

6. Common problems

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

The navigation bar

The wrong item is selected, or Items[SelectedIndex] is a group header

What you see. You set `SelectedIndex := 1` to select the first item of a group, and nothing or a different item is selected.

Why. `SelectedIndex` is an index into the flat `Items` collection, and group headers count. In demo 02, index 1 is the Sales header, which can never be selected.

Fix. Count the headers, as demo 02 does (Orders is 2), or look the index up: `oItem.Index` is the index of an item you kept from `AddItem`.

The page does not change when I select an item in code

What you see. Setting `Nav.SelectedIndex` in code, or a click on a breadcrumb segment, highlights the item, but your page stays the same.

Why. Your page switching code is in `OnItemClick`, which fires only when the user clicks an item or presses Enter on it.

Fix. Switch pages in `OnSelectionChange`, which fires for every change of the selection, whatever its cause. Keep `OnItemClick` for commands.

The first page is empty when the form opens

What you see. `SelectedIndex` is set in the Object Inspector and the item is highlighted, but your `OnSelectionChange` handler did not run, so the page is empty.

Why. `OnSelectionChange` is not fired while the form is loading from its DFM.

Fix. Fill the first page yourself in `FormCreate`, as demo 05 does with `DoFillList`, or call your `OnSelectionChange` handler from there.

A click on an item or a group header does nothing

What you see. No event fires for some rows.

Why. A disabled item (`Enabled = False`) ignores the mouse and the keyboard, like Suppliers in demo 02. A group header is never selected and never fires `OnItemClick`: in the accordion it folds its group, in the sidebar it is a label.

Fix. Enable the item, or use an ordinary item (`IsGroup = False`) for anything that must open a page.

Space does not activate an item

What you see. Enter opens the focused item, Space does not.

Why. When `ShowSearch` is on, the characters you type go to the search box, and Space is one of them.

Fix. Use Enter, which always activates the focused item.

The bar jumps to its new width instead of animating

What you see. Setting `Collapsed` changes the width at once.

Why. The animation runs only at run time and only while `AnimateCollapse` is `True`. In the form designer the width always changes at once.

Fix. Check `AnimateCollapse`; at design time nothing is wrong.

Tabs, breadcrumb and toasts

My view tabs have close buttons and a + button

What you see. A strip meant for fixed views, such as Overview and Files, shows close buttons and a + button.

Why. The defaults suit documents: every tab is `Closable`, `CloseButtons` is `tcbAlways` and `ShowNewButton` is `True`.

Fix. Set `Closable = False` on every tab and `ShowNewButton = False` on the strip, as demo 04 does. For pages designed at design time, consider a `TsgcVCLNavigationNotebook`.

The + button does nothing

What you see. Clicking + adds no tab.

Why. The strip only fires `OnNewTab`; it does not know what a new document is.

Fix. Handle `OnNewTab`: create the page control, call `Tabs.Tabs.AddTab`, set the `Control` of the new tab and its `ActiveIndex`, as demo 03 does.

A closed tab leaves its page behind

What you see. After closing tabs, memory grows, or the page of a closed tab is still around.

Why. The strip removes the tab, but it does not free the control of the tab: the control belongs to you.

Fix. Free it in `OnTabClose`: read `Control`, set the tab's `Control := nil`, then free the control, as demos 03 and 05 do. Set `aAllow := False` first when the tab must stay, for example to ask the user to save a modified document.

Undeclared identifier: 'tpsTopRight'

What you see. Code that sets `Toasts.Position` does not compile.

Why. The toast manager is declared in `sgcNavigation_Toast_VCL`, but the `TsgcToastPosition` values are declared in `sgcNavigation_Toast`. The severity values, such as `nsvInfo`, live in `sgcNavigation_Notification`.

Fix. Add `sgcNavigation_Toast` (and `sgcNavigation_Notification` for the severities) to the `uses` clause. Setting the property in the Object Inspector needs nothing extra.

The toast appears in an unexpected place, or not in a screenshot

What you see. The toast shows over the corner of the form, not of the screen, or a window capture of the form does not contain it.

Why. A toast is a small window of its own. With `Anchor = tanForm`, the default, the toasts stack in the client area of the owner form.

Fix. Set `Anchor = tanScreen` to use the work area of the monitor. To capture a toast, capture the screen rather than the form.

The standard controls do not follow the theme, or one control ignores it

What you see. After `sgcUISetApplicationTheme(utDark, True)` the sgcUI controls are dark but the panels and combo boxes stay light; or one navigation control keeps its old look.

Why. Theming the standard controls is done by `sgcUICore_Theme_VCL`, which must be linked into the application. A control whose own `Theme` is not `utDefault` is pinned and no longer follows the application theme.

Fix. Add `sgcUICore_Theme_VCL` to a `uses` clause, as demo 04 does, and pass `True` as the second parameter. Set the `Theme` of the control back to `utDefault`.

Editions and compiling

Undeclared identifier Action, or "Property Action does not exist"

What you see. Code that sets `Item.Action` fails with E2003, or a form saved with an item linked to an action raises an “Error reading” exception when it loads.

Why. The Community edition compiles without `SGC_NAVIGATION_ACTIONS`, so the `Action` property of the items does not exist.

Fix. Use the `OnClick` of the item or the `OnItemClick` event of the bar, or a Registered or Trial build.

Pinned tabs, tab reorder or the breadcrumb edit mode do nothing

What you see. `Pinned := True` has no effect and reads back `False`, the tabs cannot be dragged, and the breadcrumb path cannot be typed. The command palette and the dock manager are missing from the palette.

Why. These features belong to `SGC_NAVIGATION_ADVANCED`, which the Community edition does not define.

Fix. Use a Registered or Trial build for them. Everything in this guide works without them.

A larger demo does not compile

What you see. `SidebarApp`, `SidebarFMX` or `Accordion` stops with unknown units or properties.

Why. `SidebarApp` and `SidebarFMX` host a dashboard and need sgcDashboard installed. `Accordion` uses action links, which the Community edition does not have.

Fix. Install sgcDashboard for the sidebar demos; use a Registered or Trial build for `Accordion`. The QuickStart demos have no such dependencies.

7. Where next

You now know the four controls of a navigation shell and how they work together. The larger demos, the help and the website take it from here.

The other Delphi demos

Besides `00.QuickStart`, the `Demos\Navigation\Delphi` folder holds these demos. Each has a `readme.html` with more detail.

Folder	What it shows
`Accordion`	Groups, multi expand and action links: an accordion bar whose items are linked to the actions of a `TActionList`.
`SidebarApp`	A sidebar with icons, badges, a disabled item, a footer item and an animated collapse, driving the pages of a `TPageControl`, one of them an sgcDashboard dashboard.
`SidebarFMX`	The sidebar demo for FireMonkey, driving the tabs of a `TTabControl`.
`Notebook`	A tabbed notebook with pages designed in the form: tab colours, tabs on any edge, wrapped rows and a page that refuses to be left until it is filled in.
`NotebookFMX`	The notebook demo for FireMonkey.
`NavigationShell`	An application shell for the VCL: a ribbon, a tree with a search row, tabs that open from the tree, a breadcrumb, a command palette on Ctrl+Shift+P, light and dark themes and a right to left layout.
`NavigationShellFMX`	The application shell for FireMonkey.
`DockingShell`	A docking shell for the VCL: an Explorer tree, a Properties panel, an auto hidden Output panel, memo documents with pinned and modified tabs, a command palette and a layout saved to JSON.
`DockingShellFMX`	A docking workspace for FireMonkey, with floating panels on desktop.

Help topics

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

Topic	What you find there
Introduction	What sgcNavigation does and where to start.
Architecture	The framework neutral models, the renderers and the VCL and FireMonkey controls.
Choosing sgcNavigation	How sgcNavigation compares with the navigation controls of RAD Studio.
Editions	What each edition includes.
Sidebar mode, Accordion mode, Tree mode	The three modes of the bar in depth.
Search, Badges, Collapse, Keyboard	Filtering, counters, the rail and the keys.
Tabs, Breadcrumb, Toasts, Themes	The other controls and the look.
Troubleshooting	More problems and their causes.

On the website

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

---

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