sgcPlanner

Getting Started

Put a calendar, a Gantt chart and a task board on a Delphi form,
one small demo at a time

About this guide

This guide is for Delphi developers who know how to build a VCL form and want to add scheduling to it with sgcPlanner: a calendar of appointments, a Gantt chart of project tasks, a Kanban board of cards, or all three at once. You do not need any experience with planning components. If you have used a calendar in an e-mail program or a task board in a project tool, you already know what the controls look like; this guide shows how to drive them from Delphi.

The guide follows the five QuickStart demos. Each demo adds one idea to the previous one, from a week calendar with seven appointments to a small team planner with a calendar, a timeline and a board that share the same tasks. 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, drag a few appointments around, then read its section and try the suggested changes. Chapters 1 to 3 explain what sgcPlanner 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\Planner\Delphi\00.QuickStart` folder of your sgcUI installation. Each demo has its own subfolder, from `01.HelloScheduler` to `05.TeamPlannerApp`, with a Delphi project (`.dpr`) that you can open and run directly. All of them are VCL applications for Windows.

The demos calculate their dates from the Monday of the current week, so they always open on a filled week, and your screen shows other dates than the figures. The screenshots were taken on a computer with Spanish regional settings, which is why the day and month names read “lu. 19” or “octubre”. Dates, times and numbers always follow the Windows regional settings of the machine that runs the demo, so on your machine they appear in your own language and format.

Contents

1. What sgcPlanner is

The idea in one minute
Your first calendar needs one event handler
What you need
Part of sgcUI
Coming from grids and hand drawn calendars

2. Install and the palette

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

3. Core concepts

The data lives apart from the views
One item for appointments, tasks and cards
Resources and statuses
Colours
Feeding the data
One selection for every view
Editing: view events and data events
Undo
Themes
Building a planner in code
Summary: who does what

4. The QuickStart demos

4.1 Demo 01: Hello Scheduler
4.2 Demo 02: Scheduler Views
4.3 Demo 03: Scheduler Events
4.4 Demo 04: Gantt and Board
4.5 Demo 05: Team Planner App

5. Main components

Reference table
The data component
The three views
Sources and files
Beyond the basics

6. Common problems

Data and display
Editing
Editions and compiling

7. Where next

The other Delphi demos
Help topics
On the website
Getting help

1. What sgcPlanner is

sgcPlanner adds planning controls to Delphi applications, for the VCL and for FireMonkey. It has three views: a **Scheduler**, the calendar you know from e-mail programs, with day, week, month and agenda views; a **Gantt** chart, with a task tree on the left and one bar per task on a time axis; and a **Kanban** board, with one column per status and one card per task. The user can create, move, resize and edit items with the mouse and the keyboard in all of them.

What makes sgcPlanner different from three separate controls is that the views do not keep the data themselves. The appointments, tasks, people and statuses live in one non visual component, `TsgcPlannerData`, and each view only shows it. Point a calendar, a Gantt chart and a board at the same data component, and a task moved on the board is moved in the calendar and in the chart at the same moment, without a line of synchronisation code.

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

Imagine four tasks of a small project, each with a title, a start, a finish, a status and a person:

Title	Start	Finish	Status	Person
Requirements	Monday	Wednesday	Done	Anna
Visual design	Thursday	Friday	In progress	Carla
Backend API	Thursday	next Tuesday	In progress	Ben
Testing	next Wednesday	next Friday	To do	Ben

The same four rows answer three different questions, and each sgcPlanner view answers one of them:

- **When is what?** The Scheduler places every task in its time slot on a calendar page: a day, a work week, a week, a month or a list of the coming days.
- **How does the project run?** The Gantt chart draws one bar per task from its start to its finish, fills the bar with the progress, and links tasks that must wait for each other.
- **Where does every task stand?** The Kanban board puts each task as a card in the column of its status. Drag a card to another column and its status changes.

In sgcPlanner each row of the table is an *item* of `TsgcPlannerData.Items`. The people are its `Resources` and the statuses are its `Statuses`. Fill the data once, connect the views through their `Data` property, and every view draws the same items in its own way. That is the entire idea; everything else is detail.

Your first calendar needs one event handler

The first QuickStart demo is the smallest planner there is. The form holds two components, a `TsgcPlannerData` named `PlannerData` and a `TsgcVCLScheduler` named `Scheduler`. In the Object Inspector the Scheduler gets its `Data` and the hours to show, stored in the form file like this:

```
  object Scheduler: TsgcVCLScheduler
    Align = alClient
    Data = PlannerData
    Options.DayStartHour = 8
    Options.DayEndHour = 19
  end
  object PlannerData: TsgcPlannerData
  end
```

The appointments are added in `FormCreate`, inside a `BeginUpdate` and `EndUpdate` bracket. A small helper method creates one item and sets its title, its start, its finish and its colour:

```
procedure TfrmHelloScheduler.DoAddAppointment(const aTitle: string;
  aDay: Integer; aFromHour, aToHour: Double; aColor: Cardinal);
var
  vMonday: TDateTime;
  oItem: TsgcPlannerItem;
begin
  vMonday := StartOfTheWeek(Date);
  oItem := PlannerData.AddItem;
  oItem.Title := aTitle;
  oItem.Start := vMonday + aDay + aFromHour / 24;
  oItem.Finish := vMonday + aDay + aToHour / 24;
  oItem.Color := aColor;
end;

procedure TfrmHelloScheduler.FormCreate(Sender: TObject);
begin
  PlannerData.BeginUpdate;
  try
    DoAddAppointment('Team meeting', 0, 9, 10, $FF2A78D6);
    DoAddAppointment('Customer call', 0, 11.5, 12.5, $FFEB6834);
    // ...
    DoAddAppointment('Release', 4, 14, 16.5, $FF0891B2);
  finally
    PlannerData.EndUpdate;
  end;
end;
```

That is all. When `EndUpdate` runs, the Scheduler finds seven new items in its data, lays them out in their days and hours and paints the week once. You did not write any drawing code, any hit testing or any code to move an appointment: the user can already drag the tiles to another time and resize them.

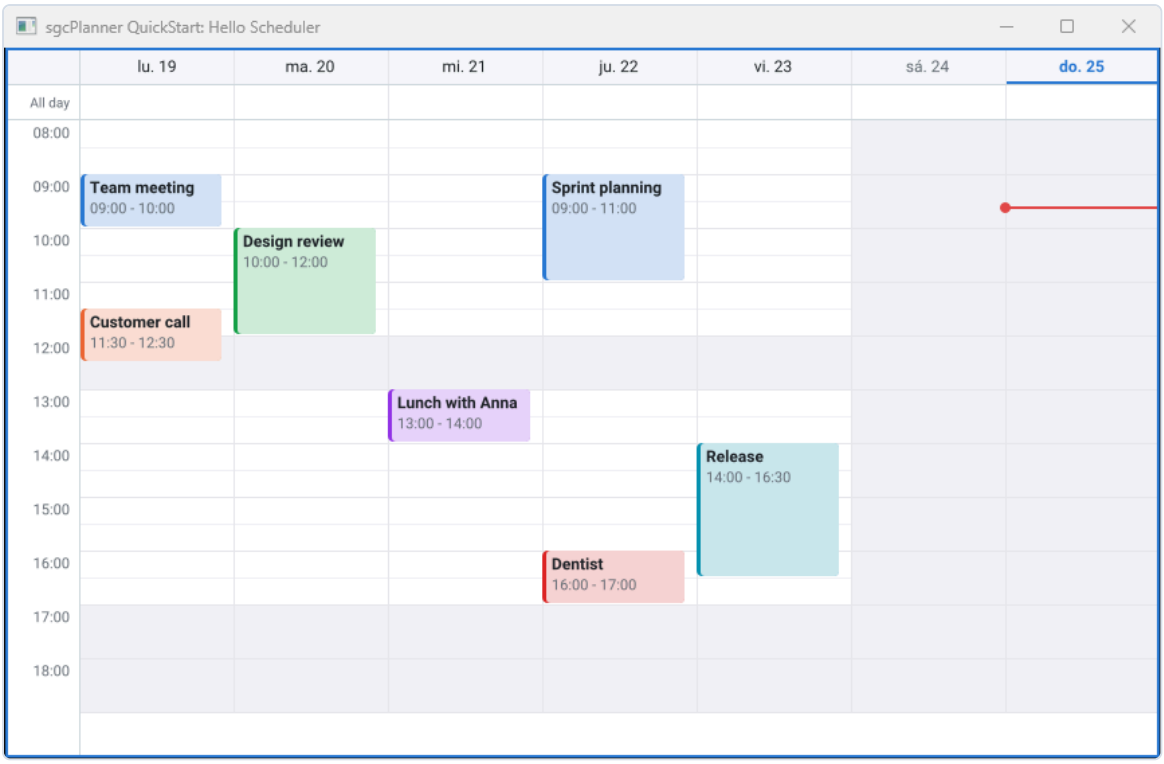


Figure 1.1. The result of demo 01: one week from 8:00 to 19:00 with seven appointments, each in its own colour.

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.** `TsgcVCLScheduler`, `TsgcVCLGantt` and `TsgcVCLKanban` run on Windows, Win32 in every version and Win64 where your Delphi version can build it.
- **FireMonkey.** `TsgcFMXScheduler`, `TsgcFMXGantt` and `TsgcFMXKanban` have their own packages from Delphi XE6 on, for the platforms FireMonkey targets in your Delphi version.
- **No window at all.** The data component and `TsgcPlannerRenderer` use no visual framework, so a console application or a server can load a plan and write each view to a PNG or SVG file.
- **sgcUICore.** The shared foundation of the sgcUI visual products. It is installed together with sgcPlanner, you do not install it separately.

The QuickStart demos are VCL applications and need nothing beyond Delphi and sgcPlanner.

Part of sgcUI

sgcPlanner is one product of sgcUI, the eSeGeCe suite of visual components (grids, charts, dashboards, pivot tables, 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 Scheduler looks the same on the VCL, on FireMonkey and in a PNG made by the renderer, and it matches the other sgcUI controls on your form. The `Theme` property (`utDefault`, `utLight`, `utDark`, `utCustom`, `utHighContrast`) works the same way on all of them.
- The colours you give to items, resources and statuses are sgcUICore colours: a `TsgcUIColor` value written as `$AARRGGBB`. Chapter 3 explains them.

Coming from grids and hand drawn calendars

Many Delphi applications show appointments in a `TDBGrid`, or draw a week calendar by hand in the `OnDrawCell` event of a `TStringGrid` or on a `TPaintBox`. That works until the user wants to drag an appointment to another day, resize it, see the same data as a timeline, or switch to a dark theme. sgcPlanner gives you those parts ready made: you describe the data, the controls do the layout, the painting and the mouse handling.

If you know data aware controls, the closest idea is the data module: the data lives in one place and every control on every form shows it. In sgcPlanner the data component plays that role, and it can be filled from your own code, from a JSON file, or from your database tables through `TsgcPlannerSource_DataSet`.

2. Install and the palette

This chapter gets sgcPlanner 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

sgcPlanner 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 **sgcPlanner**. The installer shows one checkbox per product and enables the products your licence includes. sgcUICore has no checkbox: it is installed automatically with sgcPlanner.
4. Optionally select **sgcPlanner for sgcWebSockets**, the bridge with the calendar synchronisation components (Google Calendar, Outlook, CalDAV and iCalendar feeds). It installs only when sgcWebSockets is present, and you do not need it for this guide.
5. The installer copies the files, compiles the packages on your machine for each selected Delphi version and installs the design time packages.

Everything goes into one folder under `Documents\esegece` by default. 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 sgcPlanner units without any setting on your side.

The packages

You only need this section if you build the packages yourself. sgcPlanner 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
`sgcPlanner<V>`	Runtime	The data component, the working calendar, the view models and their painters, JSON, `TsgcPlannerRenderer` and `TsgcPlannerSource_DataSet`. No VCL, no FireMonkey.
`sgcPlannerVCL<V>`	Runtime	`TsgcVCLScheduler`, `TsgcVCLGantt` and `TsgcVCLKanban`.
`sgcPlannerFMX<V>`	Runtime, XE6 and later	`TsgcFMXScheduler`, `TsgcFMXGantt` and `TsgcFMXKanban`.
`dclsgcPlanner<V>`	Design time	Registers the VCL controls and the non visual components on the palette.
`dclsgcPlannerFMX<V>`	Design time, XE6 and later	Registers the FireMonkey controls. Install `dclsgcPlanner` first.
`sgcPlannerWS<V>`, `dclsgcPlannerWS<V>`	Runtime and design time	The sgcWebSockets bridge with the calendar synchronisation components.

Build them in dependency order: sgcUICore first, then `sgcPlanner`, then `sgcPlannerVCL`, then `dclsgcPlanner`. 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 Planner**. In a VCL project it shows these components:

Component	Visual	What it does
`TsgcPlannerData`	no	The data: items, resources, statuses, dependencies, the working calendar, the selection and undo. Every view points to one.
`TsgcVCLScheduler`	yes	The calendar: day, work week, week, month and agenda views, and more.
`TsgcVCLGantt`	yes	The Gantt chart: a task tree on the left, bars on a time axis on the right.
`TsgcVCLKanban`	yes	The board: one column per status, one card per item.
`TsgcPlannerSource_DataSet`	no	Loads the data from `TDataSet` tables, follows their changes and can write the edits back.
`TsgcPlannerRenderer`	no	Paints a Scheduler, Gantt or Kanban view without a window, to a PNG or an SVG file.
`TsgcPlannerICalendar`	no	Imports and exports iCalendar (`.ics`) files. Registered editions only.
`TsgcPlannerMSProject`	no	Imports and exports Microsoft Project XML files. Registered editions only.

In a FireMonkey project the same page shows `TsgcFMXScheduler`, `TsgcFMXGantt` and `TsgcFMXKanban` instead of the VCL controls, together with the same non visual components. With the sgcWebSockets bridge installed, the calendar synchronisation components are added as well. The QuickStart demos use only the first four components of the table.

To set up a planner at design time, drop a `TsgcPlannerData` first, then a view, and pick the data component in the drop down list of the view's `Data` property. The `Resources` and `Statuses` of the data component are ordinary collections: click the ellipsis button next to them in the Object Inspector to add people and statuses with their `Id`, name or caption and colour. The appointments themselves (`Items`) are normally added from code or loaded from a file or a dataset.

Editions

sgcPlanner is built from the same source for every edition. The Registered and Trial editions include everything; the Trial edition is meant for evaluation. The free Community edition leaves out the advanced features, which are compiled under the define `SGC_PLANNER_ADVANCED`:

- the Gantt chart stores and saves the dependencies between tasks but does not draw the link arrows, and has no drag to link and no automatic scheduling, critical path or baselines;
- the resource Timeline view of the Scheduler paints a notice instead of the timeline, and recurring items are shown once, at their own start;
- the Kanban board has no swimlanes and no WIP limits;
- `TsgcPlannerICalendar` and `TsgcPlannerMSProject` do not exist.

Everything else is in every edition: the three views, the data component with resources, statuses and dependencies, drag and drop, in place editing, summaries, milestones, progress, undo, themes, the DataSet source, JSON files and the renderer.

> **Note.** The QuickStart demos avoid the advanced features, so they compile and run with every edition. The one visible difference is in demo 04: in a Community build its Gantt chart shows the bars without the arrows between linked tasks. The links are still there in the data.

Where the demos are

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

Folder	Project	What it shows
01.HelloScheduler	`HelloScheduler.dpr`	The smallest planner: a week calendar with seven appointments added in code
02.SchedulerViews	`SchedulerViews.dpr`	Day, work week, week, month and agenda views, navigation, people with their own colour
03.SchedulerEvents	`SchedulerEvents.dpr`	Creating, moving, renaming and deleting with the mouse, and the events that let you allow or refuse it
04.GanttBoard	`GanttBoard.dpr`	A Gantt chart and a Kanban board on the same project data, statuses, progress, a milestone and task links
05.TeamPlannerApp	`TeamPlannerApp.dpr`	A small team planner: calendar, timeline and board in tabs, shared selection, undo, save and open, dark theme

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

Running your first demo

1. In Delphi, choose **File** > **Open Project** and open `Demos\Planner\Delphi\00.QuickStart\01.HelloScheduler\HelloScheduler.dpr`.
2. Press **F9**. The window opens on the current week with seven appointments.
3. Drag an appointment to another day, or drag its bottom edge to make it longer. Nothing in the demo handles this: the Scheduler and the data component do it on their own.
4. Back in the IDE, open the form, select `Scheduler` and look at its `Data` and `Options` properties in the Object Inspector.

> **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

Ten ideas explain almost everything sgcPlanner does. Read them once now, and come back to them when a demo surprises you.

The data lives apart from the views

`TsgcPlannerData` is a non visual component. It paints nothing; it holds five parts:

Property	What it holds
`Items`	The appointments, tasks and cards, each a `TsgcPlannerItem`.
`Resources`	The people (or rooms, machines, vehicles) an item is assigned to.
`Statuses`	The steps of your workflow, such as To do, In progress and Done.
`Dependencies`	The links between tasks: this task can only start when that one has finished.
`Calendar`	The working calendar: work days, working hours and holidays.

The views, `TsgcVCLScheduler`, `TsgcVCLGantt` and `TsgcVCLKanban`, have a `Data` property that points to a data component. A view registers itself with its data and is told about every change, so it repaints on its own when an item is added, moved or deleted, whoever made the change: your code, the user in this view, or the user in another view. Any number of views, on any number of forms, can share one data component.

> **Tip.** Always connect a view to a `TsgcPlannerData` of your own. A view with no `Data` still works, with an internal data object of its own, but then your code and the other views cannot reach its items.

One item for appointments, tasks and cards

There is only one item class. The same `TsgcPlannerItem` is an appointment in the Scheduler, a bar in the Gantt chart and a card on the board. The properties you meet first are these:

Property	Meaning
`Id`	A unique string. `AddItem` generates one when you pass none, or you pass your own, such as a database key.
`Title`	The text of the tile, the bar or the card.
`Start`, `Finish`	Ordinary `TDateTime` values. One hour is `1 / 24` of a day.
`AllDay`	Shows the item in the all day row of the Scheduler instead of in a time slot.
`Status`	The `Id` of a status of `Statuses`. It decides the column on the board.
`Resources`	The `Id` of a resource, or several separated by commas.
`Progress`	From 0 to 100, drawn as the fill of a Gantt bar and as a bar on a card.
`Milestone`	A point in time, drawn as a diamond in the Gantt chart. Give it the same `Start` and `Finish`.
`ParentId`	The `Id` of a parent item. A parent becomes a summary task in the Gantt tree.
`Color`	An optional colour of its own.
`Notes`, `Tags`	Free text and labels.

The data component manages the items for you. `AddItem` creates one, `FindItem` returns the item with an `Id` (or `nil`), `DeleteItem` deletes one, `MoveItem` moves one to another status and position on the board, and `Clear` empties everything except the calendar.

Resources and statuses

Resources and statuses are small collections that you usually fill at design time in the Object Inspector, as demos 02, 04 and 05 do. Each entry has an `Id`, a display text (`Name` for a resource, `Caption` for a status) and a `Color`. Items refer to them only by `Id`, so you can rename “In progress” to “Doing” without touching a single item.

- **Resources** answer “who?”. In the Scheduler, `Options.GroupByResource` splits the day views into one column per resource. In the Gantt chart, `BarColorMode = pbcResource` colours the bars by person.
- **Statuses** answer “how far?”. The Kanban board builds its columns from the statuses (`ColumnSource = pkcStatuses`, the default), so a new status means a new column. In the Gantt chart, `BarColorMode = pbcStatus` colours the bars by status.

In code, `Resources.Find` and `Statuses.Find` return the entry with an `Id`, or `nil` when there is none, and `AddResource` and `AddStatus` add one.

Colours

Every colour in sgcPlanner is a `TsgcUIColor`, a `Cardinal` written as `$AARRGGBB`: alpha, red, green, blue. Use `$FF` as the alpha for a solid colour, so `$FF2A78D6` is a solid blue. This is not the byte order of a VCL `TColor`, so do not pass `clRed` or a `TColor` constant. In a DFM file the same values are stored as signed integers, which is why the form files of the demos show `Color = -13993770` for `$FF2A78D6`.

When a view paints an item, it looks for a colour in this order:

1. the `Color` of the item, when it is not 0;
2. else the colour of its status;
3. else the colour of its first resource.

So an item with no colour of its own takes the colour of its status, and an item with neither a colour nor a status takes the colour of its person. Demo 02 relies on the last rule and demo 05 on the second one. The Gantt chart can override the rule with `BarColorMode`: `pbcStatus` or `pbcResource` colour the bars by status or by person whatever the items say, and `pbcItem`, the default, uses the general rule.

Feeding the data

There are three ways to get items into a data component:

- **From code**, with `AddItem` and the item properties, as every QuickStart demo does. Wrap the loop in `BeginUpdate` and `EndUpdate`: the views and the `OnChange` event are told once, after the last item, instead of once per property of every item.
- **From a file or a string**, with `SaveToFile` and `LoadFromFile`, or `SaveToJSON` and `LoadFromJSON`. The JSON holds the items, the resources, the statuses, the dependencies and the calendar. Loading replaces the whole data in one update; a document that cannot be read raises an exception and leaves the data unchanged.
- **From a database**, with a `TsgcPlannerSource_DataSet` between your `TDataSet` and the data component. It loads the records, follows their changes and can write the edits of the user back. The `DataSetPlanner` demo listed in chapter 7 shows it.

This is the JSON round trip in code:

```
procedure TfrmSnippet.DoJSON;
var
  vJSON: string;
begin
  { the whole planner as a JSON string, and back }
  vJSON := PlannerData.SaveToJSON;
  PlannerData.LoadFromJSON(vJSON);
  PlannerData.Undo.Clear;
end;
```

One selection for every view

The selected item belongs to the data, not to a view. `PlannerData.SelectedId` is the `Id` of the selected item, and every view with `SyncSelection = True` (the default) highlights it. Click a card on the board and the same task is selected in the calendar and in the Gantt chart; set `SelectedId` in code and all of them follow. Each view raises `OnSelectionChange` when its selection changes. Demo 05 uses this to show the selected task in a status bar.

Editing: view events and data events

Editing is on by default. Every view has `ReadOnly = False` and `DragOptions = [pdoCreate, pdoMove, pdoResize, pdoProgress, pdoLink, pdoReorder]`, so the user can drag on empty time to create an item, drag an item to move it, drag an edge to resize it, drag the progress of a Gantt bar, and reorder cards. With `InplaceEdit = True`, F2 opens a small editor for the title.

Two groups of events let you react, and they are raised by different components:

Raised by	Events	Use them for
The view	`OnItemClick`, `OnItemDblClick`, `OnItemDelete`, `OnSelectionChange`; on the Scheduler also `OnItemCreate`, `OnDateChange`, `OnViewChange`; `OnBeginEdit` and `OnEndEdit` for the in place editor; on the board `OnCardMoved`	What the user does in that control: a click, a double click, the Delete key, a new item drawn with the mouse.
The data	`OnItemChanging`, `OnChange`	Every change of the data, whichever view or piece of code made it.

`OnItemChanging` is raised *before* a drag is applied. The item still has its old values, and the `aFields` parameter says which fields the drop changes (`pifStart`, `pifFinish`, `pifStatus` and so on). Set `aAccept` to `False` to refuse the change. `OnChange` is raised *after* every change; its `aChange` record tells what kind of thing changed (`pckItems`, `pckResources`, `pckSelection` and others) and, when exactly one item changed, its `ItemId`. Demo 03 uses both.

Undo

The data component records every change in `PlannerData.Undo`, which is enabled by default. All the changes made between `BeginUpdate` and `EndUpdate` form one undo step. `Undo.Undo` and `Undo.Redo` step back and forth, `Undo.CanUndo` and `Undo.CanRedo` tell whether there is anything to undo or redo, and `Undo.Clear` forgets the history.

> **Watch out.** Undo also records the items your own code adds at start up. Call `PlannerData.Undo.Clear` after loading, or the first Undo of the user removes your sample data. Demo 05 does exactly that.

Themes

Each view has a `Theme` property: `utDefault`, `utLight`, `utDark`, `utCustom` or `utHighContrast`, from the unit `sgcUICore_Theme`. The theme sets the background, the grid lines, the text and the selection colours; the colours of your items, resources and statuses stay the same. The theme belongs to each view, so to switch an application to dark you set it on every view, as demo 05 does.

Building a planner in code

Nothing in sgcPlanner needs the form designer. This `FormCreate` creates a data component with one person and two statuses, a Scheduler and a Kanban board that share it, and one task that appears in both views:

```
uses
  SysUtils, DateUtils, Classes, Controls, Forms,
  // sgc
  sgcUICore_Theme, sgcPlanner_Data, sgcPlanner_VCL, sgcPlanner_Scheduler,
  sgcPlanner_Scheduler_VCL, sgcPlanner_Kanban, sgcPlanner_Kanban_VCL;

procedure TfrmSnippet.FormCreate(Sender: TObject);
var
  oItem: TsgcPlannerItem;
begin
  { the data: one person and two statuses }
  PlannerData := TsgcPlannerData.Create(Self);
  PlannerData.Resources.AddResource('anna', 'Anna Smith').Color := $FF2A78D6;
  PlannerData.Statuses.AddStatus('todo', 'To do', $FF64748B);
  PlannerData.Statuses.AddStatus('done', 'Done', $FF16A34A);

  { two views of the same data }
  Scheduler := TsgcVCLScheduler.Create(Self);
  Scheduler.Parent := Self;
  Scheduler.Align := alLeft;
  Scheduler.Width := 500;
  Scheduler.Data := PlannerData;
  Scheduler.View := psvWorkWeek;

  Board := TsgcVCLKanban.Create(Self);
  Board.Parent := Self;
  Board.Align := alClient;
  Board.Data := PlannerData;
  Board.Theme := utDark;

  { one item, shown by both views }
  oItem := PlannerData.AddItem('t1');
  oItem.Title := 'Write the guide';
  oItem.Start := Date + 9 / 24;
  oItem.Finish := Date + 11 / 24;
  oItem.Resources := 'anna';
  oItem.Status := 'todo';
end;
```

`PlannerData`, `Scheduler` and `Board` are fields of the form class. Note the units: `sgcPlanner_Data` for the data, `sgcPlanner_VCL` for the common VCL host, and for every view its view model unit (`sgcPlanner_Scheduler`, `sgcPlanner_Kanban`, `sgcPlanner_Gantt`) plus its VCL unit (`sgcPlanner_Scheduler_VCL` and so on). The view model unit declares the types such as `psvWorkWeek`. When you drop the components on a form, Delphi adds most of these units for you.

Summary: who does what

Piece	Its job
Your code, a JSON file, or `TsgcPlannerSource_DataSet`	Puts the items, resources and statuses into the data component.
`TsgcPlannerData`	Stores everything, keeps the selection and the undo history, and tells the views and `OnChange` about every change.
`TsgcVCLScheduler`	Shows the items on a calendar and lets the user create, move and resize them.
`TsgcVCLGantt`	Shows the items as a task tree and bars on a time axis, with progress, milestones and links.
`TsgcVCLKanban`	Shows the items as cards in one column per status and changes the status when a card is dragged.

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.HelloScheduler	`HelloScheduler.dpr`	The data component, a view connected through `Data`, items added in code
02.SchedulerViews	`SchedulerViews.dpr`	View kinds, navigation, resources and their colours, one column per person
03.SchedulerEvents	`SchedulerEvents.dpr`	Editing with the mouse and the keyboard, view events, data events, refusing a change
04.GanttBoard	`GanttBoard.dpr`	The Gantt chart and the Kanban board, statuses, progress, a milestone, task links, two views on one data
05.TeamPlannerApp	`TeamPlannerApp.dpr`	Three views in tabs, shared selection, undo, save and open, themes

> **Note.** The demos need sgcPlanner installed, so the components appear on the SGC Planner palette page and the `sgcUI\Source` folder is on the library path. They use only features that are part of every edition. Their dates are calculated from the Monday of the current week, and the day and month names in the screenshots are Spanish because of the regional settings of the machine they were taken on.

4.1 Demo 01: Hello Scheduler

Folder `01.HelloScheduler`, project `HelloScheduler.dpr`.

What you will build

The smallest sgcPlanner application. A week calendar from 8:00 to 19:00 opens on the current week, with seven appointments in their own colours: a team meeting and a customer call on Monday, a design review on Tuesday, and so on until a release on Friday afternoon. You can drag them to another time and resize them right away.

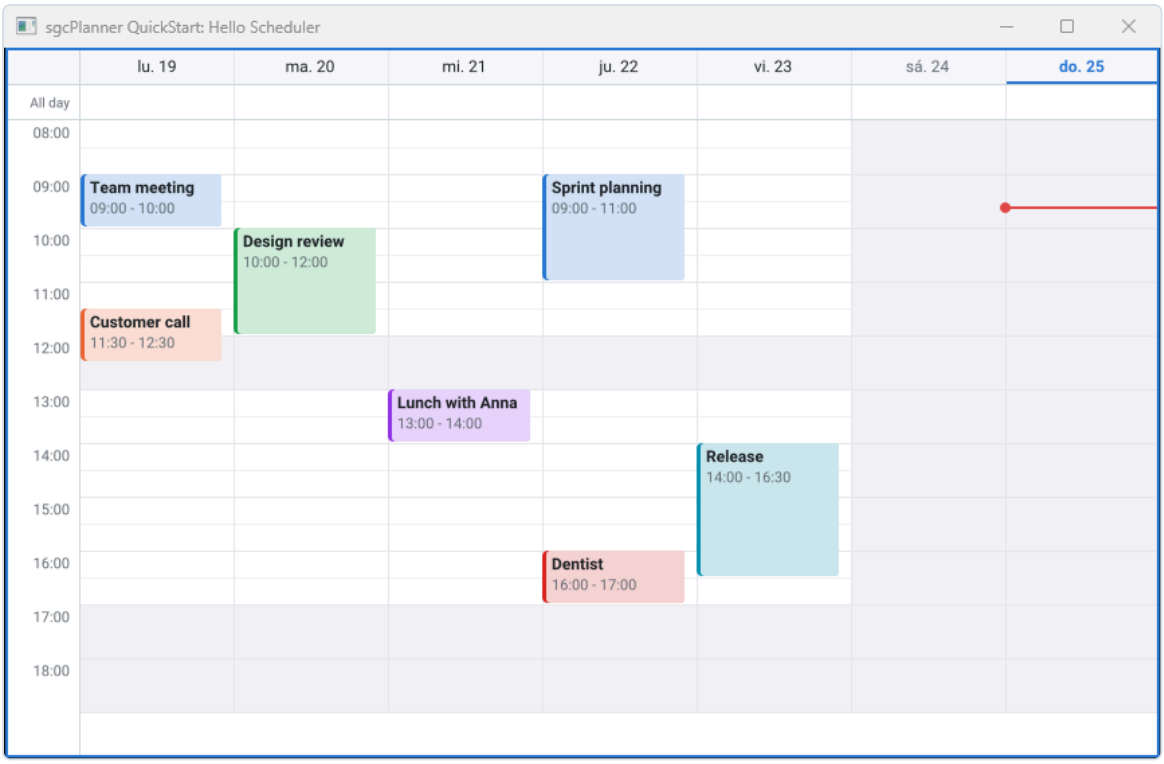


Figure 4.1. Demo 01. The week starts on Monday; the hours before 8:00 and after 19:00 are not shown because of `DayStartHour` and `DayEndHour`.

Components on the form

Component	Class	Properties you set
`PlannerData`	`TsgcPlannerData`	Nothing. It is filled from code.
`Scheduler`	`TsgcVCLScheduler`	`Align = alClient`, `Data = PlannerData`, `Options.DayStartHour = 8`, `Options.DayEndHour = 19`. `View` keeps its default, `psvWeek`.

The form file is short. `View` does not appear in it because `psvWeek` is the default value:

```
object Scheduler: TsgcVCLScheduler
  Left = 0
  Top = 0
  Width = 860
  Height = 528
  Align = alClient
  TabOrder = 0
  Data = PlannerData
  Options.DayStartHour = 8
  Options.DayEndHour = 19
end
object PlannerData: TsgcPlannerData
  Left = 40
  Top = 80
end
```

The code

The `uses` clause names the data unit, the common VCL unit and the two units of the Scheduler. `DateUtils` is there for `StartOfTheWeek`:

```
uses
  SysUtils, DateUtils, Classes, Controls, Forms,
  // sgc
  sgcPlanner_Data, sgcPlanner_VCL, sgcPlanner_Scheduler,
  sgcPlanner_Scheduler_VCL;
```

A helper method adds one appointment. The day is counted from the Monday of the current week, the hours are fractions of a day:

```
procedure TfrmHelloScheduler.DoAddAppointment(const aTitle: string;
  aDay: Integer; aFromHour, aToHour: Double; aColor: Cardinal);
var
  vMonday: TDateTime;
  oItem: TsgcPlannerItem;
begin
  { StartOfTheWeek (unit DateUtils) returns the Monday of the current week
    at 0:00. aDay 0 is Monday, 4 is Friday. One hour is 1 / 24 of a day. }
  vMonday := StartOfTheWeek(Date);

  { AddItem creates the appointment, an empty Id would generate one }
  oItem := PlannerData.AddItem;
  oItem.Title := aTitle;
  oItem.Start := vMonday + aDay + aFromHour / 24;
  oItem.Finish := vMonday + aDay + aToHour / 24;
  { colours are $AARRGGBB values, $FF is fully opaque }
  oItem.Color := aColor;
end;
```

And `FormCreate` calls it seven times inside an update bracket:

```
procedure TfrmHelloScheduler.FormCreate(Sender: TObject);
begin
  { BeginUpdate and EndUpdate bracket the changes, so the Scheduler repaints
    once, at EndUpdate, and not after every appointment. }
  PlannerData.BeginUpdate;
  try
    DoAddAppointment('Team meeting', 0, 9, 10, $FF2A78D6);
    DoAddAppointment('Customer call', 0, 11.5, 12.5, $FFEB6834);
    DoAddAppointment('Design review', 1, 10, 12, $FF16A34A);
    DoAddAppointment('Lunch with Anna', 2, 13, 14, $FF9333EA);
    DoAddAppointment('Sprint planning', 3, 9, 11, $FF2A78D6);
    DoAddAppointment('Dentist', 3, 16, 17, $FFDC2626);
    DoAddAppointment('Release', 4, 14, 16.5, $FF0891B2);
  finally
    PlannerData.EndUpdate;
  end;
end;
```

How it works

1. The form loads from its DFM. The Scheduler connects to `PlannerData` through its `Data` property and registers itself as a listener. The data is empty, so the week is empty.
2. `FormCreate` calls `BeginUpdate`. From now on the data collects changes without telling the Scheduler.
3. Each `DoAddAppointment` call creates an item with `AddItem`. No `Id` is passed, so the data generates a unique one. The start and finish are the Monday of this week plus the day plus the hour divided by 24.
4. `EndUpdate` tells the Scheduler once that the items changed. It lays out the seven appointments in their days and paints the week.

5. When the user drags a tile, the Scheduler changes the `Start` and `Finish` of the item in the data, the data notifies its listeners, and the Scheduler repaints. The demo has no code for this.

Try this

- Set `Scheduler.View` to `psvMonth` in the Object Inspector. The same seven items now appear as small entries in a month grid.
- Set `Scheduler.Options.SlotMinutes` to 15. Each hour is now divided into four slots, and drags snap to quarter hours.
- Set `Scheduler.Theme` to `utDark`.

4.2 Demo 02: Scheduler Views

Folder `02.SchedulerViews`, project `SchedulerViews.dpr`.

What you will build

A calendar with a toolbar, like the one of an e-mail program: buttons to go back, to today and forward, a label with the dates on screen, buttons for the Day, Work week, Week, Month and Agenda views, and a check box that splits the day views into one column per person. The appointments of last week, this week and next week belong to three people, Anna, Ben and Carla, and each appointment takes the colour of its person.

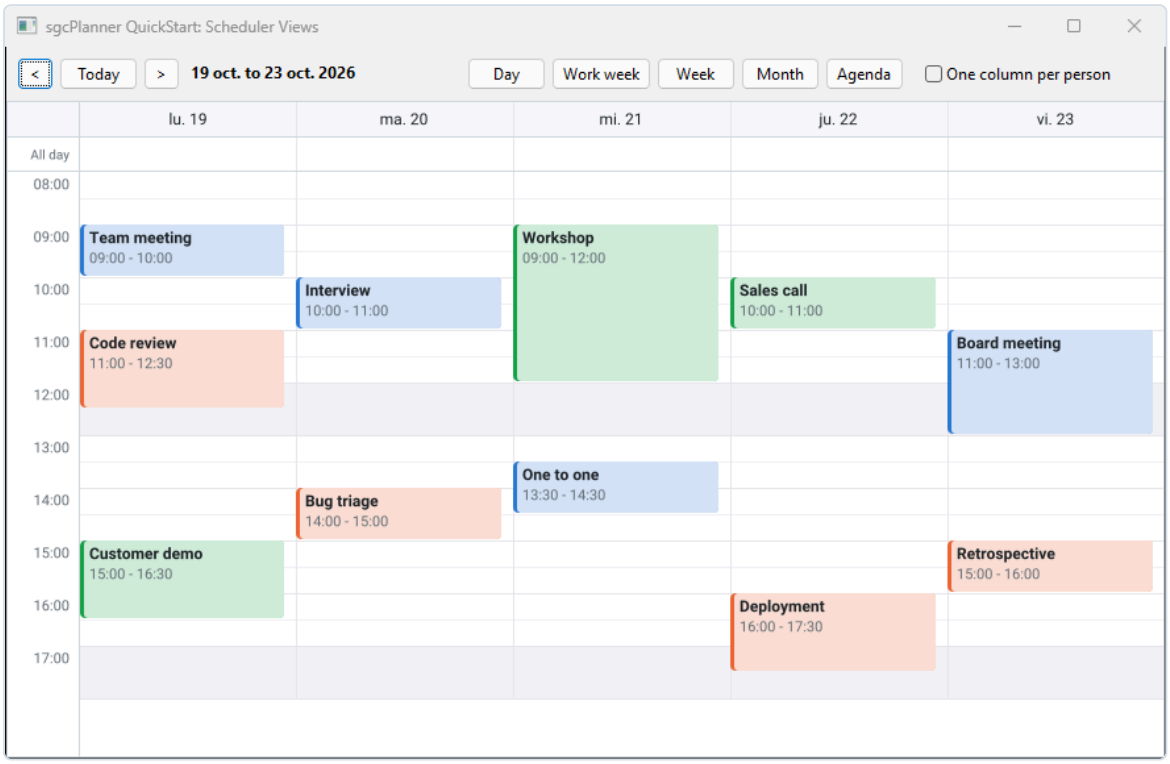


Figure 4.2. Demo 02 in the work week view. Blue is Anna, orange is Ben and green is Carla: the colours come from the resources, the items have no colour of their own.

Components on the form

Component	Class	Properties you set
`PlannerData`	`TsgcPlannerData`	`Resources`: `anna` "Anna Smith" (blue), `ben` "Ben Jones" (orange), `carla` "Carla Ruiz" (green).
`Scheduler`	`TsgcVCLScheduler`	`Align = alClient`, `Data = PlannerData`, `View = psvWorkWeek`, `Options.DayStartHour = 8`, `Options.DayEndHour = 18`, `OnDateChange` and `OnViewChange = SchedulerDateChange`.
`btnPrior`, `btnToday`, `btnNext`	`TButton`	Captions `<`, `Today`, `>`.
`btnDay` to `btnAgenda`	`TButton`	`Tag` 0 to 4, all with `OnClick = btnViewClick`.
`lblRange`	`TLabel`	Shows the dates on screen.
`chkGroup`	`TCheckBox`	Caption "One column per person".

The three people are stored with the data component in the form file. The colours are the signed integer form of `$FF2A78D6`, `$FFEB6834` and `$FF16A34A`:

```
object PlannerData: TsgcPlannerData
  Resources = <
    item
      Id = 'anna'
      Name = 'Anna Smith'
      Color = -13993770
    end
    item
      Id = 'ben'
      Name = 'Ben Jones'
      Color = -1349580
    end
    item
      Id = 'carla'
      Name = 'Carla Ruiz'
      Color = -15293622
    end>
end
```

The code

The helper now takes a person instead of a colour. The item gets the `Id` of a resource in its `Resources` property:

```
procedure TfrmSchedulerViews.DoAddAppointment(const aTitle, aPerson: string;
  aDay: Integer; aFromHour, aToHour: Double);
var
  oItem: TsgcPlannerItem;
begin
  { aDay is counted from the Monday of the current week: -7 is the Monday
    before, 7 the Monday after }
  oItem := PlannerData.AddItem;
  oItem.Title := aTitle;
  oItem.Start := StartOfTheWeek(Date) + aDay + aFromHour / 24;
  oItem.Finish := StartOfTheWeek(Date) + aDay + aToHour / 24;
  { the id of a resource of PlannerData.Resources; no Color is set, so the
    appointment takes the colour of the resource }
  oItem.Resources := aPerson;
end;

procedure TfrmSchedulerViews.FormCreate(Sender: TObject);
begin
  PlannerData.BeginUpdate;
  try
    // last week
    DoAddAppointment('Budget review', 'anna', -5, 10, 12);
    DoAddAppointment('Supplier visit', 'ben', -3, 9, 11);
    DoAddAppointment('Training', 'carla', -2, 14, 17);
    // this week
    DoAddAppointment('Team meeting', 'anna', 0, 9, 10);
    // ...
    // next week
    DoAddAppointment('Kick off', 'anna', 7, 9, 10.5);
    DoAddAppointment('Trade fair', 'carla', 8, 9, 17);
    DoAddAppointment('Release', 'ben', 10, 14, 16);
  finally
    PlannerData.EndUpdate;
  end;
  DoUpdateRange;
end;
```

The toolbar is a handful of one line handlers. The view buttons share one handler: the `Tag` of each button is the ordinal value of its view kind, so `Tag = 3` is `psvMonth`:

```
procedure TfrmSchedulerViews.btnPriorClick(Sender: TObject);
begin
  Scheduler.Prior;
end;

procedure TfrmSchedulerViews.btnTodayClick(Sender: TObject);
begin
  Scheduler.GoToday;
end;

procedure TfrmSchedulerViews.btnNextClick(Sender: TObject);
begin
  Scheduler.Next;
end;

procedure TfrmSchedulerViews.btnViewClick(Sender: TObject);
begin
  { the Tag of the button is the ordinal of its view }
  Scheduler.View := TsgcPlannerSchedulerViewKind(TButton(Sender).Tag);
end;

procedure TfrmSchedulerViews.chkGroupClick(Sender: TObject);
begin
  Scheduler.Options.GroupByResource := chkGroup.Checked;
end;
```

The label shows the dates on screen. `VisibleRange` returns the first visible day and the day *after* the last one, and the handler runs from both `OnDateChange` and `OnViewChange`:

```
procedure TfrmSchedulerViews.DoUpdateRange;
var
  vStart, vFinish: TDateTime;
begin
  { the finish of VisibleRange is exclusive, the last visible day is the
    day before it }
  Scheduler.VisibleRange(vStart, vFinish);
  if vFinish - vStart <= 1 then
    lblRange.Caption := FormatDateTime('dddd d mmmm yyyy', vStart)
  else
    lblRange.Caption := FormatDateTime('d mmm', vStart) + ' to ' +
      FormatDateTime('d mmm yyyy', vFinish - 1);
end;

procedure TfrmSchedulerViews.SchedulerDateChange(Sender: TObject);
begin
  { assigned to both OnDateChange and OnViewChange in the Object
    Inspector }
  DoUpdateRange;
end;
```

How it works

1. The view kinds are the values of `TsgcPlannerSchedulerViewKind`: `psvDay`, `psvWorkWeek`, `psvWeek`, `psvMonth`, `psvAgenda`, and two more, `psvTimeline` and `psvYear`. The demo offers the first five. The Timeline view is a Registered edition feature.
2. `Prior`, `Next` and `GoToday` move the date of the Scheduler by one period of the current view, for example one week in the work week view. Every move raises `OnDateChange`, every change of `View` raises `OnViewChange`, and both update the label.

3. No item has a `Color`, and none has a status, so the colour rule of chapter 3 falls through to the first resource of each item.
4. `Options.GroupByResource` splits the day, work week and week views into one column per resource, so you can compare the agendas of the three people side by side.

Try this

- Click Day, then check “One column per person”. Anna, Ben and Carla each get a column.
- Give one appointment its own colour with `oItem.Color := $FF9333EA`. The item colour wins over the colour of the person.
- Set `Scheduler.Options.AgendaDays` to 14, run the demo and click Agenda. The list now covers two weeks.

4.3 Demo 03: Scheduler Events

Folder `03.SchedulerEvents`, project `SchedulerEvents.dpr`.

What you will build

A work week calendar with a "What happened" log on the right. Every gesture of the user is written to the log: a new appointment drawn with the mouse, a click, a move, a rename, a delete. One appointment, "Lunch (locked)", cannot be moved, resized or deleted, and the log says why.

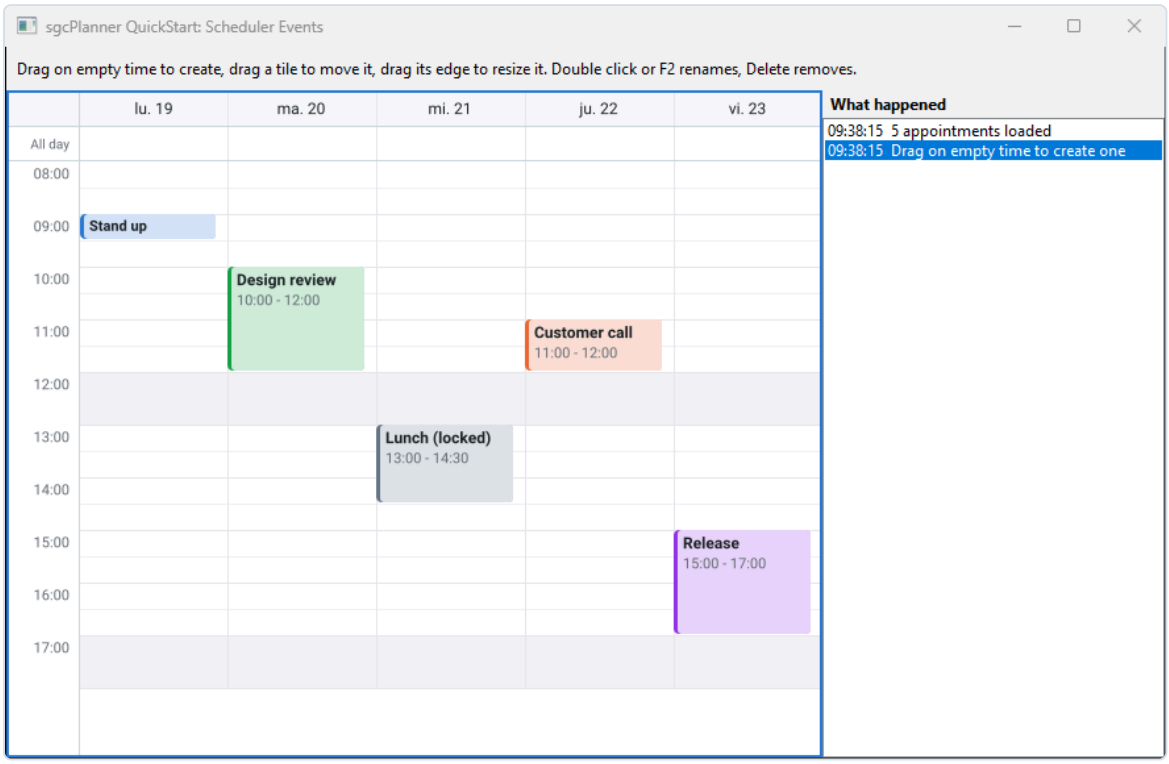


Figure 4.3. Demo 03. The log on the right lists what the user did and what the demo allowed or refused.

Components on the form

Component	Class	Properties you set
`PlannerData`	`TsgcPlannerData`	`OnChange = PlannerDataChange`, `OnItemChanging = PlannerDataItemChanging`.
`Scheduler`	`TsgcVCLScheduler`	`Data = PlannerData`, `View = psvWorkWeek`, hours 8 to 18, `InplaceEdit = True`, `OnItemCreate`, `OnItemClick`, `OnItemDblClick`, `OnItemDelete`, `OnEndEdit`.
`lstLog`	`TListBox`	The log.
`lblHelp`	`TLabel`	The help line on top.

The events are assigned in the Object Inspector. Note which ones belong to the Scheduler and which to the data:

```
  object Scheduler: TsgcVCLScheduler
    Align = alClient
    Data = PlannerData
    InplaceEdit = True
    View = psvWorkWeek
    Options.DayStartHour = 8
    Options.DayEndHour = 18
    OnItemClick = SchedulerItemClick
    OnItemDblClick = SchedulerItemDblClick
    OnEndEdit = SchedulerEndEdit
    OnItemDelete = SchedulerItemDelete
    OnItemCreate = SchedulerItemCreate
  end
  object PlannerData: TsgcPlannerData
    OnChange = PlannerDataChange
    OnItemChanging = PlannerDataItemChanging
  end
```

The code

The appointments get their own `Id` this time, so the code can recognise the locked lunch by `Id` later:

```
procedure TfrmSchedulerEvents.DoAddAppointment(const aId, aTitle: string;
  aDay: Integer; aFromHour, aToHour: Double; aColor: Cardinal);
var
  oItem: TsgcPlannerItem;
begin
  oItem := PlannerData.AddItem(aId);
  oItem.Title := aTitle;
  oItem.Start := StartOfTheWeek(Date) + aDay + aFromHour / 24;
  oItem.Finish := StartOfTheWeek(Date) + aDay + aToHour / 24;
  oItem.Color := aColor;
end;

procedure TfrmSchedulerEvents.FormCreate(Sender: TObject);
begin
  PlannerData.BeginUpdate;
  try
    DoAddAppointment('standup', 'Stand up', 0, 9, 9.5, $FF2A78D6);
    DoAddAppointment('review', 'Design review', 1, 10, 12, $FF16A34A);
    DoAddAppointment('lunch', 'Lunch (locked)', 2, 13, 14.5,
      $FF64748B);
    DoAddAppointment('call', 'Customer call', 3, 11, 12, $FFEB6834);
    DoAddAppointment('release', 'Release', 4, 15, 17, $FF9333EA);
  finally
    PlannerData.EndUpdate;
  end;
  DoLog(Format('%d appointments loaded', [PlannerData.Items.Count]));
  DoLog('Drag on empty time to create one');
end;
```

The view events. `OnItemCreate` names the new appointment, `OnItemDblClick` renames one with a dialog, `OnItemDelete` refuses to delete the lunch and asks for confirmation otherwise, and `OnEndEdit` refuses an empty title from the in place editor:

```
procedure TfrmSchedulerEvents.SchedulerItemCreate(Sender: TObject;
  const aStart, aFinish: TDateTime; const aResourceId: string;
  var aTitle: string; var aAccept: Boolean);
begin
  aTitle := 'New appointment';
  aAccept := True;
  DoLog('Created on ' + FormatDateTime('ddd hh:nn', aStart));
end;

{ OnItemDblClick: rename the appointment with a dialog }
procedure TfrmSchedulerEvents.SchedulerItemDblClick(Sender: TObject;
  aItem: TsgcPlannerItem);
var
  vTitle: string;
begin
  vTitle := aItem.Title;
  if InputQuery('Rename', 'Title:', vTitle) then
    aItem.Title := vTitle;
end;

{ OnItemDelete: the user pressed Delete on the selected appointment }
procedure TfrmSchedulerEvents.SchedulerItemDelete(Sender: TObject;
  aItem: TsgcPlannerItem; var aAccept: Boolean);
begin
  if aItem.Id = 'lunch' then
  begin
    aAccept := False;
    DoLog('The company lunch cannot be deleted');
  end
  else
    aAccept := MessageDlg('Delete "' + aItem.Title + '"?', mtConfirmation,
      [mbYes, mbNo], 0) = mrYes;
end;

// ...

{ OnEndEdit: the in place editor (F2) closes. aText is the new title, it
  can be changed here, and aAccept False keeps the old one. }
procedure TfrmSchedulerEvents.SchedulerEndEdit(Sender: TObject;
  aItem: TsgcPlannerItem; var aText: string; var aAccept: Boolean);
begin
  aText := Trim(aText);
  aAccept := aText <> '';
end;
```

The data events. `OnItemChanging` locks the lunch against moves and resizes, whichever view tries it. `OnChange` logs every change of one item; when `FindItem` returns `nil`, the item was deleted:

```
{ OnItemChanging: raised once per drop, before the change is applied. The
  item still has its old values; aFields says what the drop changes. }
procedure TfrmSchedulerEvents.PlannerDataItemChanging(Sender: TObject;
  aItem: TsgcPlannerItem; const aFields: TsgcPlannerItemFields;
  var aAccept: Boolean);
begin
  if (aItem.Id = 'lunch') and ((pifStart in aFields) or (pifFinish in aFields))
  then
  begin
    aAccept := False;
    DoLog('The company lunch is locked');
  end;
end;

{ OnChange: raised after every change. aChange.Kinds says what changed and
  aChange.ItemId is the item, when exactly one item changed. }
procedure TfrmSchedulerEvents.PlannerDataChange(Sender: TObject;
  const aChange: TsgcPlannerChange);
var
  oItem: TsgcPlannerItem;
begin
  if (pckItems in aChange.Kinds) and (aChange.ItemId <> '') then
  begin
    oItem := PlannerData.FindItem(aChange.ItemId);
    if oItem = nil then
      DoLog('Deleted ' + aChange.ItemId)
    else
      DoLog('"' + oItem.Title + '" ' + DoTimes(oItem));
  end;
end;
```

How it works

These are the gestures the Scheduler understands out of the box, and the event each one raises:

The user	The Scheduler	Event
drags on empty time	creates an item for the dragged slots	`OnItemCreate` (view), then `OnChange` (data)
drags a tile	moves it	`OnItemChanging`, then `OnChange`
drags the top or bottom edge	resizes it	`OnItemChanging`, then `OnChange`
clicks a tile	selects it	`OnItemClick`
double clicks a tile	nothing on its own	`OnItemDblClick`
presses F2, or has just drawn a new item	opens the in place editor (with `InplaceEdit = True`)	`OnEndEdit` when it closes
presses Delete	deletes the selected item	`OnItemDelete`, then `OnChange`

1. Events that end in `var aAccept: Boolean` can refuse what the user is doing: set `aAccept` to `False` and the Scheduler puts everything back.
2. The lock is in `OnItemChanging` of the data, not in a Scheduler event. That is deliberate: if you add a Gantt chart to this form later, the lunch stays locked there as well, without another line of code.
3. `OnItemChanging` receives the item with its *old* values and the set of fields the drop is about to change. It does not receive the new values. Check the fields, not the dates.

4. `OnChange` runs after the change, also for changes made in code, such as the rename in `SchedulerItemDblClick`. That is why the log shows the new title after a rename without any extra code.

Try this

- Set `Scheduler.ReadOnly` to `True`. The user can still look and select, but can no longer drag, create or delete appointments.
- Remove `pdoResize` from `Scheduler.DragOptions`. Appointments can still be moved, but not resized.
- Set `Scheduler.Options.SlotMinutes` to 15 and create a few appointments. They snap to quarter hours.

4.4 Demo 04: Gantt and Board

Folder `04.GanttBoard`, project `GanttBoard.dpr`.

What you will build

A small "website relaunch" project shown two ways at once. On top, a Gantt chart with seven tasks and a "Go live" milestone: bars coloured by status, filled by their progress, a red line for today, and arrows from each task to the tasks that wait for it. Below, a Kanban board with the columns To do, In progress, Review and Done. Drag a card to another column and the bar above changes colour; drag a bar to other dates and the due date on the card follows.



Figure 4.4. Demo 04. Both views show the same eight items. The board shows the finish of each item as a due date, in red when it is overdue.

Components on the form

Component	Class	Properties you set
`PlannerData`	`TsgcPlannerData`	`Statuses`: `todo` "To do" (grey), `doing` "In progress" (blue), `review` "Review" (amber), `done` "Done" (green).
`Gantt`	`TsgcVCLGantt`	`Align = alTop`, `Height = 290`, `Data = PlannerData`, `TreeWidth = 240`, `BarColorMode = pbcStatus`.
`Splitter`	`TSplitter`	`Align = alTop`, between the two views.
`Board`	`TsgcVCLKanban`	`Align = alClient`, `Data = PlannerData`, `Columns` with `Width = 200` for each status, `ShowUnknownColumn = False`, `OnCardMoved = BoardCardMoved`.
`pnlStatus`	`TPanel`	A status line at the bottom.

The board needs no list of columns to work: with `ColumnSource = pkcStatuses`, the default, it makes one column per status. The `Columns` collection only adjusts those columns, matched by `StatusId`; here it makes them narrower than the default so four fit in the window:

```
  object Gantt: TsgcVCLGantt
    Align = alTop
    Data = PlannerData
    TreeWidth = 240
    BarColorMode = pbcStatus
  end
  // ...
  object Board: TsgcVCLKanban
    Align = alClient
    Data = PlannerData
    Columns = <
      item
        StatusId = 'todo'
        Width = 200
      end
      // ...
      item
        StatusId = 'done'
        Width = 200
      end>
    ShowUnknownColumn = False
    OnCardMoved = BoardCardMoved
  end
```

The code

The `uses` clause now names the two units of the Gantt chart and the two of the board:

```
uses
  SysUtils, DateUtils, Classes, Controls, Forms, ExtCtrls,
  // sgc
  sgcPlanner_Data, sgcPlanner_VCL, sgcPlanner_Gantt, sgcPlanner_Gantt_VCL,
  sgcPlanner_Kanban, sgcPlanner_Kanban_VCL;
```

A task has a status and a progress. A task runs from 8:00 of its first day to 17:00 of its last day:

```
procedure TfrmGanttBoard.DoAddTask(const aId, aTitle, aStatus: string;
  aFirstDay, aLastDay: Integer; aProgress: Double);
var
  oItem: TsgcPlannerItem;
begin
  { the days are counted from the Monday of the current week, a task runs
    from 8:00 of its first day to 17:00 of its last day }
  oItem := PlannerData.AddItem(aId);
  oItem.Title := aTitle;
  oItem.Start := StartOfTheWeek(Date) + aFirstDay + 8 / 24;
  oItem.Finish := StartOfTheWeek(Date) + aLastDay + 17 / 24;
  oItem.Status := aStatus;
  oItem.Progress := aProgress;
end;
```

`FormCreate` adds the seven tasks, the milestone and the links between them:

```
procedure TfrmGanttBoard.FormCreate(Sender: TObject);
var
  oItem: TsgcPlannerItem;
begin
  PlannerData.BeginUpdate;
  try
    DoAddTask('t1', 'Requirements', 'done', -7, -5, 100);
    DoAddTask('t2', 'Wireframes', 'done', -4, -3, 100);
    DoAddTask('t3', 'Visual design', 'review', 0, 2, 90);
    DoAddTask('t4', 'Backend API', 'doing', 0, 2, 60);
    DoAddTask('t5', 'Frontend', 'doing', 3, 8, 20);
    DoAddTask('t6', 'Content', 'todo', 7, 8, 0);
    DoAddTask('t7', 'Testing', 'todo', 9, 11, 0);

    { a milestone is a point in time: Start = Finish and Milestone = True }
    oItem := PlannerData.AddItem('t8');
    oItem.Title := 'Go live';
    oItem.Start := StartOfTheWeek(Date) + 14 + 9 / 24;
    oItem.Finish := oItem.Start;
    oItem.Milestone := True;
    oItem.Status := 'todo';

    { finish to start links: the second task starts after the first one }
    PlannerData.Dependencies.AddDependency('t1', 't2');
    PlannerData.Dependencies.AddDependency('t2', 't3');
    PlannerData.Dependencies.AddDependency('t3', 't5');
    PlannerData.Dependencies.AddDependency('t4', 't5');
    PlannerData.Dependencies.AddDependency('t5', 't7');
    PlannerData.Dependencies.AddDependency('t6', 't7');
    PlannerData.Dependencies.AddDependency('t7', 't8');
  finally
    PlannerData.EndUpdate;
  end;
end;
```

When the form is shown, the Gantt chart zooms its time axis so the whole project fits. When a card is dropped, the demo completes the progress of a finished task and writes the new status in the status line:

```
procedure TfrmGanttBoard.FormShow(Sender: TObject);
begin
  { zoom the time axis so the whole project fits in the chart }
  Gantt.ZoomToFit;
end;

{ OnCardMoved: a card was dropped in another column (its Status is already
  the new one) or at another position of the same column. }
procedure TfrmGanttBoard.BoardCardMoved(Sender: TObject;
  aItem: TsgcPlannerItem);
begin
  if aItem.Status = 'done' then
    aItem.Progress := 100;
  pnlStatus.Caption := '  "' + aItem.Title + '" is now ' +
    PlannerData.Statuses.Find(aItem.Status).Caption;
end;
```

How it works

1. The same items feed both views. The Gantt chart uses `Start`, `Finish`, `Progress` and `Milestone`; the board uses `Status` to pick the column, `Finish` for the due date badge and `Progress` for the bar on the card.
2. With `BarColorMode = pbcStatus`, every bar takes the colour of its status, the same colour as the top border of its column on the board.
3. `AddDependency(FromId, ToId)` creates a finish to start link: the second task waits for the first to finish. The optional third and fourth parameters choose another kind (`pdkSS`, `pdkFF`, `pdkSF`) and a lag. The Gantt chart draws each link as an arrow, in the Registered and Trial editions.
4. Dragging a card to another column sets the `Status` of the item. The data notifies both views, so the bar in the Gantt chart takes the colour of the new status at once. Then `OnCardMoved` runs, and the demo sets the progress to 100 for a finished task, which also fills the bar above.
5. `ShowUnknownColumn = False` hides the extra column the board otherwise shows for items whose status matches no column.

> **Note.** `ZoomToFit` picks the first of the fixed zoom levels at which the whole project fits. With a project of about three and a half weeks in this window it chooses the week scale, which leaves some empty space on the right of the chart. That is the expected behaviour, not a layout problem.

Try this

- Set `Gantt.BarColorMode` to `pbcItem` and give a few tasks their own `Color` in `DoAddTask`. Tasks with a colour show it; the others still fall back to the colour of their status.
- Add a task “Design” with `Id 'd'` and set the `ParentId` of `t2` and `t3` to `'d'`. The Gantt tree now shows a summary row with two children; `Gantt.CollapseAll` and `Gantt.ExpandAll` fold and unfold it.
- Set `Board.CardOptions.ShowProgress` to `False`. The progress bars disappear from the cards, not from the Gantt chart.

4.5 Demo 05: Team Planner App

Folder `05.TeamPlannerApp`, project `TeamPlannerApp.dpr`.

What you will build

A small but complete team planner. A toolbar with New task, Delete, Undo, Save, Open and a Dark theme check box; three tabs, Calendar, Timeline and Board, with the same twelve tasks of the week; and a status bar with the selected task and a count of the finished ones. Select a task on any tab and it is selected on the other two; delete it and it is gone from all three; press Undo and it comes back everywhere.

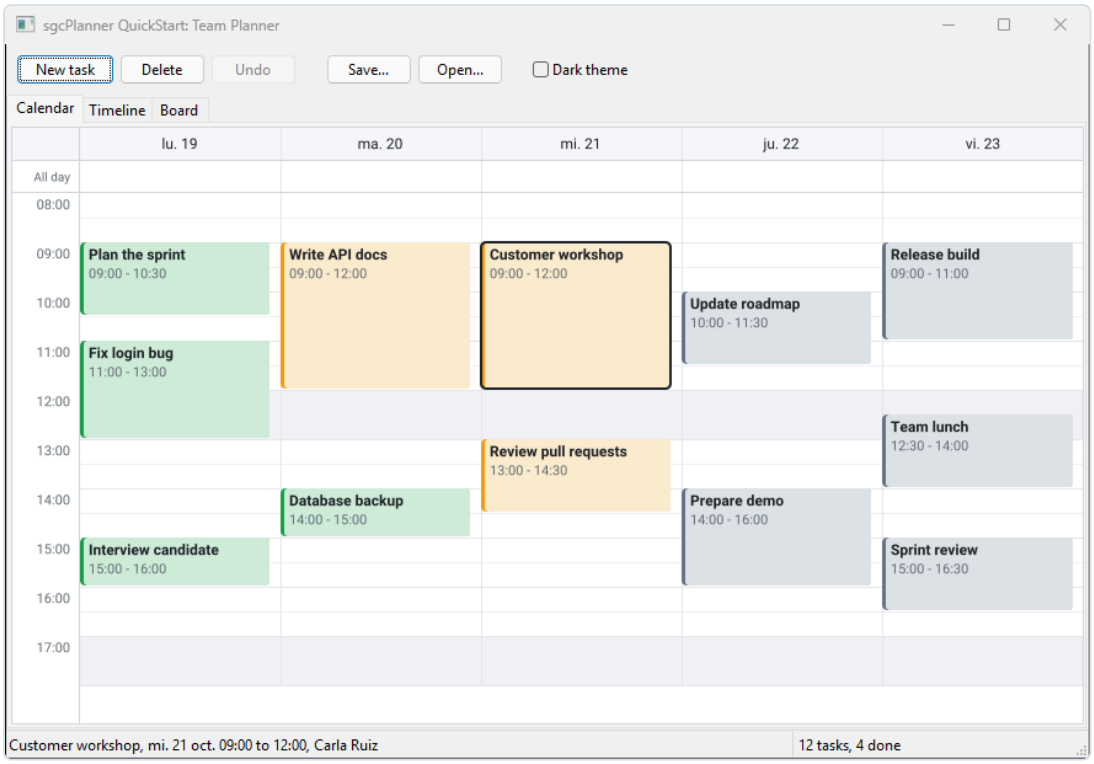


Figure 4.5. Demo 05 on the Calendar tab. The tiles take the colour of their status: green for done, amber for in progress, grey for to do. The selected task, Customer workshop, is described in the status bar.

Components on the form

Component	Class	Properties you set
`PlannerData`	`TsgcPlannerData`	`Resources` anna, ben, carla; `Statuses` todo (grey), doing (amber), done (green); `OnChange = PlannerDataChange`.
`Scheduler`	`TsgcVCLScheduler`	On the Calendar tab. `Data = PlannerData`, `View = psvWorkWeek`, hours 8 to 18, `OnSelectionChange = ViewSelectionChange`.
`Gantt`	`TsgcVCLGantt`	On the Timeline tab. `Data = PlannerData`, `TreeWidth = 240`, `BarColorMode = pbcResource`, `OnSelectionChange = ViewSelectionChange`.
`Board`	`TsgcVCLKanban`	On the Board tab. `Data = PlannerData`, `Columns` 270 pixels wide, `ShowUnknownColumn = False`, `OnSelectionChange = ViewSelectionChange`.
`PageControl`	`TPageControl`	Three tabs.
`StatusBar`	`TStatusBar`	Two panels.
`SaveDialog`, `OpenDialog`	`TSaveDialog`, `TOpenDialog`	`Filter = 'Planner files (*.json)|*.json'`, `DefaultExt = 'json'`.

The code

The `uses` clause has every unit seen so far, plus `sgcUICore_Theme` for the theme values:

```
uses
  SysUtils, DateUtils, Classes, Controls, Forms, StdCtrls, ExtCtrls,
  ComCtrls, Dialogs,
  // sgc
  sgcUICore_Theme, sgcPlanner_Data, sgcPlanner_VCL, sgcPlanner_Scheduler,
  sgcPlanner_Scheduler_VCL, sgcPlanner_Gantt, sgcPlanner_Gantt_VCL,
  sgcPlanner_Kanban, sgcPlanner_Kanban_VCL;
```

`FormCreate` adds twelve tasks with a person and a status, then clears the undo history and selects one task:

```
procedure TfrmTeamPlannerApp.FormCreate(Sender: TObject);
begin
  PlannerData.BeginUpdate;
  try
    DoAddTask('Plan the sprint', 'anna', 'done', 0, 9, 10.5);
    DoAddTask('Fix login bug', 'ben', 'done', 0, 11, 13);
    // ...
    DoAddTask('Sprint review', 'carla', 'todo', 4, 15, 16.5);
  finally
    PlannerData.EndUpdate;
  end;
  { the loading above is not something the user should undo }
  PlannerData.Undo.Clear;
  { select the customer workshop, in every view at once }
  PlannerData.SelectedId := PlannerData.Items[5].Id;
  DoUpdateStatus;
end;
```

`DoUpdateStatus` fills the status bar from the data and enables the buttons. It runs after every change of the data and every change of the selection:

```
procedure TfrmTeamPlannerApp.DoUpdateStatus;
var
  oItem: TsgcPlannerItem;
  oResource: TsgcPlannerResource;
  vPerson: string;
  vDone, i: Integer;
begin
  oItem := PlannerData.FindItem(PlannerData.SelectedId);
  if oItem = nil then
    StatusBar.Panels[0].Text := 'Nothing selected'
  else
  begin
    { Resources.Find returns nil for an unknown id }
    oResource := PlannerData.Resources.Find(oItem.Resources);
    if oResource <> nil then
      vPerson := oResource.Name
    else
      vPerson := 'nobody';
    StatusBar.Panels[0].Text := Format('%s, %s %s to %s, %s',
      [oItem.Title, FormatDateTime('ddd d mmm', oItem.Start),
      FormatDateTime('hh:nn', oItem.Start), FormatDateTime('hh:nn',
      oItem.Finish), vPerson]);
  end;

  vDone := 0;
  for i := 0 to PlannerData.Items.Count - 1 do
    if PlannerData.Items[i].Status = 'done' then
      Inc(vDone);
  StatusBar.Panels[1].Text := Format('%d tasks, %d done',
    [PlannerData.Items.Count, vDone]);

  btnDelete.Enabled := oItem <> nil;
  btnUndo.Enabled := PlannerData.Undo.CanUndo;
end;

procedure TfrmTeamPlannerApp.PlannerDataChange(Sender: TObject;
  const aChange: TsgcPlannerChange);
begin
  { every change of the data, made by a view, by the buttons or by Undo }
  DoUpdateStatus;
end;

procedure TfrmTeamPlannerApp.ViewSelectionChange(Sender: TObject);
begin
  DoUpdateStatus;
end;
```

The toolbar works on the data only. None of these handlers touches a view, except to scroll the calendar to a new task:

```
procedure TfrmTeamPlannerApp.btnNewClick(Sender: TObject);
var
  oItem: TsgcPlannerItem;
begin
  { a one hour task at the next full hour, selected so the user sees it }
  oItem := PlannerData.AddItem;
  oItem.Title := 'New task';
  oItem.Start := Date + (HourOf(Now) + 1) / 24;
  oItem.Finish := oItem.Start + 1 / 24;
  oItem.Resources := 'anna';
  oItem.Status := 'todo';
  PlannerData.SelectedId := oItem.Id;
  Scheduler.ScrollToTime(oItem.Start);
end;

procedure TfrmTeamPlannerApp.btnDeleteClick(Sender: TObject);
begin
  if PlannerData.SelectedId <> '' then
    PlannerData.DeleteItem(PlannerData.SelectedId);
end;

procedure TfrmTeamPlannerApp.btnUndoClick(Sender: TObject);
begin
  PlannerData.Undo.Undo;
end;

procedure TfrmTeamPlannerApp.btnSaveClick(Sender: TObject);
begin
  if SaveDialog.Execute then
    PlannerData.SaveToFile(SaveDialog.FileName);
end;

procedure TfrmTeamPlannerApp.btnOpenClick(Sender: TObject);
begin
  if OpenDialog.Execute then
  begin
    PlannerData.LoadFromFile(OpenDialog.FileName);
    PlannerData.Undo.Clear;
  end;
end;
```

The theme is a property of each view, so the check box sets it on all three:

```
procedure TfrmTeamPlannerApp.chkDarkClick(Sender: TObject);
var
  vTheme: TsgcUITheme;
begin
  if chkDark.Checked then
    vTheme := utDark
  else
    vTheme := utLight;
  Scheduler.Theme := vTheme;
  Gantt.Theme := vTheme;
  Board.Theme := vTheme;
end;
```

How it works

1. **Shared selection.** The selection is `PlannerData.SelectedId`. Setting it in `FormCreate` selects the workshop in the three views at once, and a click in any view changes it for all of them. Each view raises `OnSelectionChange`, which refreshes the status bar.
2. **One place for every change.** `PlannerDataChange` runs after a drag in a view, a click on New task or Delete, an Undo and a file load alike. Because the status bar is updated there, it is always right, whoever changed the data.
3. **Undo.** Every change after `FormCreate` is recorded. `Undo.Clear` after the seed data and after `LoadFromFile` makes sure the user cannot undo the load itself. `Undo.CanUndo` enables the button.
4. **Colours.** The tasks have a status and a person but no colour. On the calendar and the board the status wins, following the rule of chapter 3. The Gantt chart has `BarColorMode = pbcResource`, so on the Timeline tab the same tasks are coloured by person. One data, two colour schemes.
5. **Files.** `SaveToFile` writes the items, resources, statuses, dependencies and calendar as JSON, and `LoadFromFile` reads them back in one update.

Try this

- Save the plan, open the JSON file in a text editor, change the title of a task, and open the file again in the demo.
- Set `InplaceEdit` to `True` on the three views. F2 now renames the selected task in whichever tab is active.
- Add a fourth status, for example `review` "Review", to `PlannerData.Statuses`. The board gets a fourth column without any code. (Add a matching item to `Board.Columns` if you want it 270 pixels wide like the others.)

5. Main components

The SGC Planner palette page is short, and four of its components cover a typical first application: the data component and the three views. This chapter introduces each of them, then the components that load and save data, and finally the components you will want later.

Every class has its own help topic, named after the class, such as `TsgcVCLScheduler.htm` or `TsgcPlannerData.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
`TsgcPlannerData`	SGC Planner	The shared data of every view	`Items`, `Resources`, `Statuses`, `Dependencies`, `AddItem`, `FindItem`, `DeleteItem`, `SelectedId`, `Undo`, `OnChange`, `OnItemChanging`
`TsgcVCLScheduler`	SGC Planner	The VCL calendar	`Data`, `View`, `Options`, `Prior`, `Next`, `GoToday`, `VisibleRange`, `OnItemCreate`, `OnDateChange`
`TsgcVCLGantt`	SGC Planner	The VCL Gantt chart	`Data`, `TreeWidth`, `BarColorMode`, `Columns`, `ZoomToFit`, `ExpandAll`, `CollapseAll`
`TsgcVCLKanban`	SGC Planner	The VCL board	`Data`, `Columns`, `ColumnSource`, `ShowUnknownColumn`, `CardOptions`, `OnCardMoved`
`TsgcFMXScheduler`, `TsgcFMXGantt`, `TsgcFMXKanban`	SGC Planner (FireMonkey)	The FireMonkey views	The same as the VCL views
`TsgcPlannerSource_DataSet`	SGC Planner	Loads the data from `TDataSet` tables	`Data`, `DataSet`, `ResourceDataSet`, `DependencyDataSet`, `ItemFields`, `Active`, `LiveUpdates`, `WriteBack`
`TsgcPlannerRenderer`	SGC Planner	Paints a view without a window	`Data`, `View`, `Width`, `Height`, `Theme`, `SaveToPNG`, `SaveToSVG`
`TsgcPlannerICalendar`	SGC Planner	iCalendar import and export (Registered editions)	See its help topic
`TsgcPlannerMSProject`	SGC Planner	Microsoft Project XML import and export (Registered editions)	See its help topic
`TsgcPlannerItem`	none, an item of `Items`	An appointment, task or card	`Id`, `Title`, `Start`, `Finish`, `Status`, `Resources`, `Progress`, `Milestone`, `ParentId`, `Color`
`TsgcPlannerResource`	none, an item of `Resources`	A person or another resource	`Id`, `Name`, `Color`

Class	Palette page	What it is for	Key members
`TsgcPlannerStatus`	none, an item of `Statuses`	A step of the workflow, a column of the board	`Id`, `Caption`, `Color`

The data component

TsgcPlannerData

The component you drop first, and the one your code talks to most. It holds the items, resources, statuses, dependencies and working calendar, the selection (`SelectedId`) and the undo history (`Undo`). It notifies every connected view after each change, and raises `OnItemChanging` before a drag is applied and `OnChange` after every change. `BeginUpdate` and `EndUpdate` turn a batch of changes into one notification and one undo step. `SaveToFile`, `LoadFromFile`, `SaveToJSON` and `LoadFromJSON` save and load the whole data.

The three views

TsgcVCLScheduler

The calendar. `View` chooses the layout: `psvDay`, `psvWorkWeek`, `psvWeek` (the default), `psvMonth`, `psvAgenda`, `psvYear`, and the resource timeline `psvTimeline` in the Registered editions. `Options` holds the settings of the views, such as `DayStartHour` and `DayEndHour` (0 and 24 by default), `SlotMinutes`, `AgendaDays` and `GroupByResource`. `Prior`, `Next` and `GoToday` navigate, `VisibleRange` returns the dates on screen and `ScrollToTime` scrolls a time into view. `OnItemCreate` is raised when the user draws a new item on empty time.

TsgcVCLGantt

The Gantt chart. On the left, a task tree with the columns of `Columns`, as wide as `TreeWidth` (550 pixels by default, often too wide for a small window). On the right, a bar per task on a time axis, filled by its `Progress`, a diamond per milestone, a line for today and, in the Registered editions, the arrows of the dependencies. Items with children become summary rows that `ExpandAll` and `CollapseAll` fold and unfold. `BarColorMode` colours the bars by item, status or resource, and `ZoomToFit` zooms the time axis to the whole project.

TsgcVCLKanban

The board. With `ColumnSource = pkcStatuses`, the default, it shows one column per status and puts every item in the column of its `Status`; dragging a card to another column changes the status. `Columns` adjusts the columns by `StatusId` (width, caption, colour, collapsed). `ShowUnknownColumn`, `True` by default, adds a column for items whose status matches no column. `CardOptions` chooses what a card shows, such as the progress bar. `OnCardMoved` is raised after a card is dropped.

The three views share a common set of members: `Data`, `Theme`, `ReadOnly`, `DragOptions`, `InplaceEdit`, `SyncSelection`, and the events `OnItemClick`, `OnItemDblClick`, `OnItemDelete`, `OnSelectionChange`, `OnBeginEdit` and `OnEndEdit`.

Sources and files

TsgcPlannerSource_DataSet

Connects a data component to your database. `DataSet` is the table of items, and `ResourceDataSet` and `DependencyDataSet` are optional tables of people and links. `ItemFields` and the other field maps tell which field holds the title, the start, the finish and so on, when your field names differ from the defaults. With `LiveUpdates` the planner follows the changes of the datasets, and with `WriteBack` the edits of the user are written back to them.

JSON files

The simplest way to store a plan is the JSON format of the data component itself, as demo 05 does. It needs no extra component and works in every edition.

TsgcPlannerICalendar and TsgcPlannerMSProject

Import and export the iCalendar format used by Google Calendar, Outlook and other calendar programs, and the XML format of Microsoft Project. Both are part of the Registered editions.

Beyond the basics

- **TsgcPlannerRenderer** paints a Scheduler, Gantt or Kanban view of a data component to a PNG or SVG file, without a form, for reports, e-mails or a web server. The `HeadlessReport` demo shows it in a console application.
- **The FireMonkey views** have the same properties and events as the VCL ones and connect to the same `TsgcPlannerData`. The `PlannerFMX` demo shows the three of them.
- **The working calendar**, `PlannerData.Calendar`, holds the work days, the working hours and the holidays. The views shade non working time, and the scheduling engine of the Registered editions counts only working time.
- **The sgcWebSockets bridge** adds components that keep a data component in sync with Google Calendar, Outlook (through Microsoft Graph), a CalDAV server or an iCalendar feed. They need sgcWebSockets.

6. Common problems

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

Data and display

The Scheduler shows no appointments

What you see. The calendar is empty, although your code added items.

Why. Usually the items are not in the period on screen. The Scheduler opens on the current date, so items of another week are simply off screen. The other common cause is that the view is not connected to the data component your code fills: a view with no `Data` uses an internal data object of its own.

Fix. Check the `Data` property of the view. Calculate your dates from today, as the demos do with `StartOfTheWeek(Date)`, or move the view to the items with `Next`, `Prior` or its `Date` property. In the debugger, `PlannerData.Items.Count` tells you whether the items arrived at all.

The day starts at midnight and every appointment is tiny

What you see. The day and week views show all 24 hours, and the working day is a small band in the middle.

Why. `Options.DayStartHour` and `Options.DayEndHour` are 0 and 24 by default.

Fix. Set them to your working hours, for example 8 and 18, as demos 02 to 05 do. An appointment outside these hours is drawn clipped at the edge.

The colours are wrong or the items are invisible

What you see. An item has another colour than the one you set, or it is transparent.

Why. sgcPlanner colours are `$AARRGGBB` values, not VCL `TColor` values. A `TColor` such as `clRed` has the bytes in another order and no alpha byte, so it does not give the colour you expect. And an item without a `Color` takes the colour of its status first, then of its person, which may not be the one you expect.

Fix. Write colours with an `$FF` alpha, such as `$FFDC2626` for red. Remember the order of the rule: item, status, resource. In the Gantt chart, check `BarColorMode`.

The board shows an extra column, or a card is missing

What you see. The board has a column you did not define, or an item does not appear on the board at all.

Why. The board puts each item in the column of its `Status`. An item whose status matches no column goes to an extra column for unknown statuses, which `ShowUnknownColumn` shows by default. With `ShowUnknownColumn = False`, such an item is not shown on the board.

Fix. Give every item the `Id` of an existing status, and check the spelling: `'Done'` and `'done'` are different ids. Hide the unknown column only when every item has a valid status.

The Gantt tree takes half of the window

What you see. The task tree on the left of the Gantt chart is very wide and the bars have little room.

Why. `TreeWidth` is 550 pixels by default, and the Kanban columns are 272 pixels wide by default.

Fix. Set `TreeWidth` to a smaller value, such as 240 in demos 04 and 05, and give the board columns a `Width` in `Columns` when you want more of them on screen.

Editing

A drag does nothing or the item jumps back

What you see. The user drags an item and nothing happens, or it returns to its old place on release.

Why. The view is `ReadOnly`, the gesture is not in `DragOptions`, or one of your handlers refused the change: `OnItemChanging` of the data, `OnItemCreate` of the Scheduler, or another event with an `aAccept` parameter.

Fix. Check `ReadOnly` and `DragOptions` of the view, and set a breakpoint in the handlers that can set `aAccept` to `False`. Demo 03 shows the lock pattern on purpose.

OnItemChanging shows the old dates

What you see. In `OnItemChanging` you read `aItem.Start` to check the new start, but it still has the old value.

Why. `OnItemChanging` runs before the change. It receives the item with its current values and the set of fields that will change, not the new values.

Fix. Decide from `aFields` and from the current values. To react to the new values, use `OnChange`, which runs after the change.

The first Undo removes the sample data

What you see. The user presses Undo once and the items your code added at start up disappear.

Why. `Undo` is enabled by default and records every change, including your own loading code. A `BeginUpdate` and `EndUpdate` bracket makes the whole load one step, which is exactly the step that is undone.

Fix. Call `PlannerData.Undo.Clear` after loading, and again after `LoadFromFile`, as demo 05 does.

F2 does nothing

What you see. Pressing F2 on a selected item does not open the editor.

Why. The in place editor is off by default: `InplaceEdit` is `False`.

Fix. Set `InplaceEdit` to `True` on the view, as demo 03 does, and handle `OnEndEdit` if you want to check the new title.

An exception about OnChange changing the data for more than 8 rounds

What you see. An `EsgcPlannerError` says that `OnChange` kept changing the planner data.

Why. Your `OnChange` handler changes the data on every notification, so each change causes another one and they never settle.

Fix. Change the data in `OnChange` only when the value differs from the one you want, or do the change somewhere else, for example in `OnCardMoved` as demo 04 does.

Editions and compiling

The Gantt chart draws no arrows between linked tasks

What you see. Demo 04, or your own Gantt chart, shows the bars but no dependency arrows.

Why. The arrows are a feature of the Registered and Trial editions. In a Community build the links are stored and saved, but not drawn. In the other editions, a link is drawn only when both of its tasks are in visible rows, so a link into a collapsed summary is hidden, and a link with an unknown task id is never drawn.

Fix. Use the Registered or Trial edition to see the arrows. Check that both ids of each link exist and that their rows are expanded.

The Timeline view shows a notice

What you see. With `View = psvTimeline` the Scheduler paints a centred notice instead of a timeline.

Why. The resource timeline is part of the Registered editions.

Fix. Use one of the other views in the Community edition, for example the day view with `Options.GroupByResource`, which also shows one column per person.

A form does not load in a Community build

What you see. Opening or running a form raises “Class TsgcPlannerICalendar not found” (or `TsgcPlannerMSProject`).

Why. These two components do not exist in the Community edition.

Fix. If a project must also compile against Community, do not drop them on the form; create them in code inside `{$IFDEF SGC_PLANNER_ADVANCED}`.

Undeclared identifier psvWorkWeek, pbcStatus or utDark

What you see. The compiler does not know a view kind, a colour mode or a theme value, although the control is on the form.

Why. The types live in the view model units and in sgcUICore, not in the VCL unit that Delphi adds for the control.

Fix. Add `sgcPlanner_Scheduler` for the Scheduler types, `sgcPlanner_Gantt` for the Gantt types, `sgcPlanner_Kanban` for the board types and `sgcUICore_Theme` for the themes, as the demos do.

7. Where next

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

The other Delphi demos

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

Demo	What it shows
`ProjectPlanner`	A website relaunch project shown as a Gantt chart, a Kanban board and a week Scheduler on one data component. The natural next step after demo 05.
`TeamScheduler`	Three weeks of team meetings in every Scheduler view, drag to create, in place editing, undo and redo, reminders, time zones, recurrence and resource groups.
`ProjectGantt`	A software release plan in a Gantt chart: a three level task tree, links drawn with the mouse, automatic scheduling, the critical path, a baseline and Microsoft Project XML.
`TaskBoard`	A product backlog on a Kanban board with status columns, swimlanes, WIP limits, keyboard navigation and a collapsible Done column.
`WorkshopSchedule`	The job board of a repair shop in a work week view, with tile icons, multi line titles, footers, flag borders and a holiday.
`DataSetPlanner`	A Gantt chart and a board bound to three `TClientDataSet` tables through `TsgcPlannerSource_DataSet`, with custom field names, live updates and write back.
`ICalendarSync`	A week Scheduler that exports its items to an `.ics` file and imports calendars from other applications.
`HeadlessReport`	A console application that loads a project and writes a PNG and an SVG of each view with `TsgcPlannerRenderer`.
`PlannerFMX`	A Gantt chart, a Scheduler and a board for FireMonkey on one data component.
`GoogleCalendarSync`, `OutlookCalendarSync`, `CalDAVSync`, `ICalSubscription`	Two way synchronisation with Google Calendar, Outlook and CalDAV servers, and subscription to iCalendar feeds, through the sgcWebSockets bridge.

Some of these demos use features or products that are not part of every edition, such as automatic scheduling, swimlanes, iCalendar, Microsoft Project XML or sgcWebSockets. Their readme files say what they need.

Help topics

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

Topic	What it covers
Introduction	What sgcPlanner does and where to start.
Architecture	The data, the engines, the view models, the controls and the change notifications.
Platforms, Editions	The Delphi versions, the target platforms, the packages, and what each edition includes.
Shared data, Synchronised views	One data component behind many views.
Drag and drop, In place editing, Undo and redo	Editing in depth.
Dependencies, Working calendar, Themes	Links between tasks, working time and the look of the views.
JSON, DataSet binding	Saving and loading, and connecting to a database.
Troubleshooting	More problems and their causes.
One topic per class	`TsgcPlannerData`, `TsgcVCLScheduler`, `TsgcVCLGantt`, `TsgcVCLKanban`, `TsgcPlannerSource_DataSet`, `TsgcPlannerRenderer` and the others, with every property, its type and its default.

On the website

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

Enjoy planning.

---

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