sgcReports

Getting Started

Design banded reports in the Object Inspector, fill them from your datasets and export them to PDF, 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 print reports from their application with sgcReports: a product list, a sales summary with totals, an invoice. You do not need to know any other report tool. If you have used a banded report generator before, you will recognise the ideas at once; if you have not, chapter 1 explains them in a minute.

The guide follows the five QuickStart demos. Each demo adds one idea to the previous one, from a single page with a heading to a small invoicing application with master and detail data, totals, VAT, a PDF file and printing. Chapter 4 walks through them one at a time: what you will build, the components on the form, the report layout, the code, how it works and a few changes to try.

Read it with Delphi open next to it. Open a demo, run it, page through the preview, then read its section and try the suggested changes. Chapters 1 to 3 explain what sgcReports 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\Reports\Delphi\00.QuickStart` folder of your sgcUI installation. Each demo has its own subfolder, from `01.HelloReport` to `05.ReportInvoiceApp`, 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 system with Spanish regional settings, so they show amounts such as `1.445,00` and dates such as `04 ene.`. Numbers and dates in a report follow the Windows regional settings of the machine that runs the demo, so on your machine you may see `1,445.00` and `04 Jan`.

Contents

1. What sgcReports is

The idea in one minute
Your first report needs two lines of code
What you need
Part of sgcUI, built on sgcPDF
Coming from other report tools

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

Definition, engine, output
Pages, bands and items
Units, positions and colours
Styles
Expressions in square brackets
Data sources
Groups and aggregates
Variables and events
Designed in the form, or built in code
Prepared pages and threads

4. The QuickStart demos

4.1 Demo 01: Hello Report
4.2 Demo 02: Report Data List
4.3 Demo 03: Report Group Totals
4.4 Demo 04: Report Export Print
4.5 Demo 05: Report Invoice App

5. Main components

Reference table
The report definition
The engine
The data sources
The preview
The export components
Beyond the basics

6. Common problems

Expressions
Data and layout
Output, editions and compiling

7. Where next

The other Delphi demos
Help topics
On the website
Getting help

1. What sgcReports is

sgcReports is a report engine for Delphi. You describe a report once (its pages, the bands that repeat on them and the texts, lines, shapes, images and barcodes inside the bands), hand it your data, and the engine lays out the finished pages. The pages can be shown in a preview on a form, printed, or saved as PDF, Excel, CSV or PNG files.

You drop three components on a form: a `TsgcReport` that holds the definition, a `TsgcReportEngine` that lays it out, and a `TsgcReportPreviewVCL` that shows the result. The whole definition can be edited in the Object Inspector and is saved in the form file like any other property, so a simple report needs almost no code.

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

The idea in one minute

Think of a printed product list. At the top of the first page there is a title. Under it, a row of column captions. Then one line per product, as many as there are. At the bottom of every page there is a page number. When the list is longer than a page, the column captions appear again at the top of the next page.

A banded report describes exactly that, without knowing how many products there will be:

- **A band** is a horizontal strip of the page with a height and a type. The type says when the band prints: once at the start (report title), at the top or the bottom of every page (page header, page footer), above the data (column header), once per record (master data), once per group (group header and footer) or once at the end (summary).
- **An item** is something inside a band: a text, a line, a shape, an image, a barcode. It has a position and a size relative to its band.
- **An expression** is the part of a text between square brackets. `[Products.Price]` prints the price of the current record, `[Sum(Sales.Amount)]` prints a total, `Page [Page] of [TotalPages]` prints the page number.

The engine walks through the data, prints each band when its turn comes, starts a new page when the current one is full and repeats the headers and footers. That is the entire idea; everything else is detail.

Your first report needs two lines of code

The first QuickStart demo is the smallest report there is: a heading, a subtitle, a line, three lines of text and a page footer. The definition is set at design time, in the `Pages` collection of the `TsgcReport` component. In the form file a band and its item look like this:

```
  object Report: TsgcReport
    Pages = <
      item
        Name = 'Page1'
        // ...
        Bands = <
          // ...
          item
            Name = 'Footer'
            BandType = rbtPageFooter
            Height = 8.000000000000000000
            Items = <
              item
                Top = 2.000000000000000000
                Width = 170.000000000000000000
                Height = 5.000000000000000000
                StyleName = 'Small'
                Text.Text = 'Page [Page] of [TotalPages]'
              end>
          end>
      end>
    // ...
  end
  object Engine: TsgcReportEngine
    Report = Report
  end
```

The preview on the form has `Engine = Engine`. The only code of the demo is this event handler:

```
procedure TfrmHelloReport.FormCreate(Sender: TObject);
begin
  // lays out the report and shows the pages in the preview
  Preview.Prepare;
  // zoom so the width of the page fills the preview
  Preview.FitWidth;
end;
```

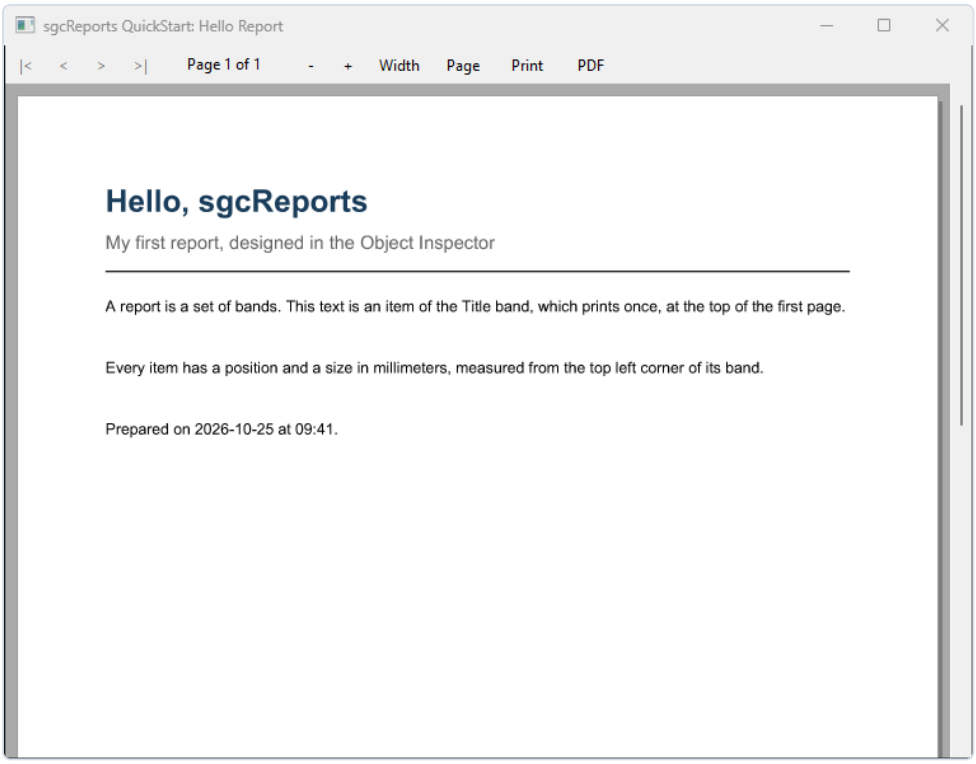


Figure 1.1. The result of demo 01: one A4 page, designed entirely in the Object Inspector.

> **Tip.** In the code and form excerpts of this guide, a line `// ...` marks lines that were left out to keep the excerpt short. The demo files have them all.

What you need

- **Delphi.** Delphi 7, 2007, 2009, 2010, XE to XE8, 10 Seattle to 10.4 Sydney, 11, 12 or 13. There is one set of packages per version.
- **A framework.** The engine itself has no VCL or FireMonkey dependency. The VCL preview, `TsgcReportPreviewVCL`, is available for every Delphi version; the FireMonkey preview, `TsgcReportPreviewFMX`, for XE5 and later. This guide uses the VCL.
- **Platforms.** The QuickStart demos are Windows VCL applications for Win32 and Win64. The engine and the exports also run in Windows services, Linux64 console applications and, through FireMonkey, on macOS, iOS and Android.
- **sgcPDF.** sgcReports is built on sgcPDF, which writes the PDF files and draws the preview. The sgcUI installer installs it together with sgcReports, so you do not have to do anything extra.
- **No DLLs.** Reports, PDF files and the exports are produced in Delphi code, with no external library and no printer driver needed to create a PDF.

Part of sgcUI, built on sgcPDF

sgcReports is one of the sgcUI products, the family of visual Delphi components by eSeGeCe that also contains sgcPDF, sgcCharts, sgcGrid, sgcPivotGrid and others. All of them are installed by the same sgcUI installer, into the same folder, with the same packages layout.

Two relationships matter for sgcReports:

- **sgcPDF** is required. The runtime package of sgcReports requires the sgcPDF package, the PDF export uses the sgcPDF writer, and the preview converts the prepared pages to a PDF and shows it in the sgcPDF viewer control. That is why the preview toolbar has a **PDF** button: the file it saves is the document it is already showing.
- **sgcUICore** is used by the end user report designer and the design time package. The report itself, the engine and the preview do not need it, and a report has no sgcUICore theme: its look comes from its own styles.

With the Registered editions, a report can also hold a chart from sgcCharts or a pivot table from sgcPivotGrid. Chapter 5 mentions these bridges; the QuickStart does not use them.

Coming from other report tools

If you have used a banded report generator in Delphi, such as QuickReport, FastReport or ReportBuilder, the vocabulary is the same: report title, page header, column header, master data, detail data, group header and footer, summary, page footer. A few things work differently in sgcReports:

- **The definition is a component.** Pages, bands, items and styles are published collections of `TsgcReport`. They are edited in the Object Inspector and stream with the form, so there is no separate report file to deploy. The same definition can also be saved to and loaded from JSON, or built in code.
- **The engine is separate from the screen.** `TsgcReportEngine` produces a set of prepared pages without any window. The preview, the printer and every export only read those pages, so the same report can run in a desktop application, a Windows service or a Linux daemon.
- **Data comes through sources.** A report does not point at a `TDataSource`. Each dataset is wrapped in a `TsgcReportDataSetSource` with a `SourceName`, registered with the engine in code, and read by expressions such as `[Products.Price]`. JSON data and data delivered by events work the same way.
- **PDF is built in.** The PDF export is tagged for accessibility (PDF/UA) by default and can also write archival PDF/A.

The `MigrateFromQuickReport` demo, listed in chapter 7, rebuilds a typical QuickReport report with sgcReports and names the QuickReport class that each band and item replaces.

2. Install and the palette

This chapter gets sgcReports 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

sgcReports 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 **sgcReports (installs sgcPDF)**. The installer shows one checkbox per product and enables the products your licence includes. Selecting sgcReports also selects sgcPDF, because sgcReports is built on it.
4. Optionally select **sgcReports HTML for sgcWebSockets** under the sgcWebSockets bridges. It adds an HTTP endpoint, a browser viewer and a web designer, 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, in dependency order, and installs the design time packages.

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

The packages

You only need this section if you build the packages yourself. sgcReports is split into a framework neutral core 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
`sgcReports<V>`	Runtime	The report model, expressions, data sources, layout, engine and the PDF, XLSX, CSV and PNG exports. Requires `rtl`, `dbrtl` and `sgcPDF<V>`. No VCL, no FireMonkey.
`sgcReportsVCL<V>`	Runtime	`TsgcReportPreviewVCL`. Requires `sgcPDFVCL<V>` and `sgcReports<V>`.
`sgcReportsFMX<V>`	Runtime, XE5 and later	`TsgcReportPreviewFMX`.
`sgcReportsDesigner<V>`, `sgcReportsDesignerVCL<V>`, `sgcReportsDesignerFMX<V>`	Runtime	The end user report designer (Registered editions). These packages use sgcUICore.
`dclsgcReports<V>`	Design time	Registers the components on the palette, with the **Preview...** command of the engine.
`dclsgcReportsFMX<V>`	Design time, XE5 and later	Registers the FireMonkey preview.
`sgcReportsCharts<V>`, `sgcReportsPivotGrid<V>`, `sgcReportsHTML<V>`	Runtime, with a design package each	The bridges to sgcCharts, sgcPivotGrid and sgcWebSockets.

Build them in dependency order: sgcPDF and sgcPDFVCL first, then `sgcReports`, then `sgcReportsVCL`, then the design time package. Then add `sgcUI\Source` to the library path of your Delphi version (**Tools** > **Options** > **Language** > **Delphi Options** > **Library**).

> **Note.** An application that uses sgcReports needs sgcPDF too, whether it is linked into the executable or deployed as runtime packages. With runtime packages, deploy `sgcPDF<V>` and `sgcPDFVCL<V>` next to `sgcReports<V>` and `sgcReportsVCL<V>`. The QuickStart demos are linked without runtime packages, so they are single executables.

The palette page

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

Component	Visual	What it does
`TsgcReport`	no	The report definition: pages, bands, items and styles.
`TsgcReportEngine`	no	Lays a report out into prepared pages, with its data sources and variables.
`TsgcReportDataSetSource`	no	Gives the report the records of any `TDataSet`.
`TsgcReportJSONSource`	no	Gives the report the records of a JSON array of objects.
`TsgcReportEventSource`	no	Gives the report records that your event handlers supply one at a time.
`TsgcReportPreviewVCL`	yes	Shows the prepared pages with a toolbar to page, zoom, print and save a PDF.
`TsgcReportPDFExport`	no	Saves the pages as a PDF file, accessible (PDF/UA) by default, optionally archival (PDF/A).
`TsgcReportXLSXExport`	no	Saves the pages as an Excel workbook.
`TsgcReportCSVExport`	no	Saves the pages as a CSV file.
`TsgcReportImageExport`	no	Saves every page as a PNG image.
`TsgcVCLReportDesigner`	yes	An end user report designer for your application. Registered editions only.

In a FireMonkey project the page also shows `TsgcReportPreviewFMX` (and `TsgcFMXReportDesigner` in the Registered editions). The QuickStart demos use the first two components, the dataset source, the VCL preview and the four export components.

Help at design time

Two design time helpers are worth knowing from the first day:

- **The Preview... command.** Right click a `TsgcReportEngine` on the form and choose **Preview...**. A window opens with the report laid out by that engine, without running the application. This works for a report without data bands, such as the one of demo 01. Data sources are registered with the engine in code, at run time, so for a report with a master data band the design time preview stops with the message `Data source "Products" not found` (with the name of your source); run the application to see that report. In the Registered editions the same menu also has **Design report...**, which opens the visual report designer.
- **The collection editors.** Select the report, click the ellipsis button of `Pages`, select a page and click the ellipsis button of its `Bands`, then select a band and open its `Items`. Each editor lists the items of one collection; select one to edit it in the Object Inspector. `Styles` has its own editor on the report.

> **Tip.** To see a whole report definition at once, right click the form in the designer and choose **View as Text**. The form excerpts in this guide come from that view, and for a report it is often the quickest way to compare positions or copy a band. Choose **View as Form** to go back to the designer.

Editions

sgcReports 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 leaves out the advanced features: the end user report designer, the DOCX export and the bridges for charts, pivot tables and the web (sgcReportsCharts, sgcReportsPivotGrid, sgcReportsHTML).

Everything else is in every edition: the report model, expressions, data sources, the engine, the previews, the PDF, PDF/A, PDF/UA, XLSX, CSV, PNG and HTML exports, rich text, watermarks, crosstabs, scripting and JSON definitions. A report saved with the Community edition opens unchanged in the Registered edition.

> **Note.** The QuickStart demos use only features that are part of every edition, so they compile and run with all of them. Chapter 7 says which of the larger demos need the Registered editions.

Where the demos are

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

Folder	Project	What it shows
01.HelloReport	`HelloReport.dpr`	The smallest report: one page, two bands, styles and an expression
02.ReportDataList	`ReportDataList.dpr`	A product list from a `TClientDataSet`: column header, master data band, number formats
03.ReportGroupTotals	`ReportGroupTotals.dpr`	Sales grouped by region with subtotals, a grand total, alternating rows, conditional formatting and variables
04.ReportExportPrint	`ReportExportPrint.dpr`	Pages prepared once, saved as PDF, Excel, CSV and PNG, and printed
05.ReportInvoiceApp	`ReportInvoiceApp.dpr`	A small invoicing application: master and detail data, one invoice per page, totals and VAT, PDF and printing

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

Running your first demo

1. In Delphi, choose **File** > **Open Project** and open `Demos\Reports\Delphi\00.QuickStart\01.HelloReport\HelloReport.dpr`.
2. Press **F9**. The window opens with the report already laid out in the preview.
3. Try the preview toolbar: the arrows page through the report, the minus and plus buttons zoom out and in, **Width** and **Page** fit the page width or the whole page, **Print** prints and **PDF** saves a PDF file.
4. Back in the IDE, open the form, select `Report` and open `Pages` in the Object Inspector. The page, its two bands and their items are the whole report.

3. Core concepts

These are the ideas you need before the demos make complete sense. Each one is short; together they explain almost everything the QuickStart does.

Definition, engine, output

sgcReports separates three jobs, and each job is a different component:

Job	Component	What it knows
Describe the report	`TsgcReport`	Pages, bands, items, styles and the title. No data, and it draws nothing.
Lay it out	`TsgcReportEngine`	Which report (`Engine.Report`), which data sources, which variables. `Engine.Prepare` walks the data and produces the finished pages.
Show or save the pages	`TsgcReportPreviewVCL` , the export components	Only the prepared pages. They do not read the data again.

The preview connects to an engine through `Preview.Engine` . `Preview.Prepare` asks the engine to prepare the report and shows the result. That is why a whole demo can run on the two lines `Preview.Prepare; Preview.FitWidth;` .

Pages, bands and items

The definition is a tree of collections, all of them published so they can be edited in the Object Inspector:

- `Report.Pages` holds the pages. A page (`TsgcReportPage`) has a `PaperSize` (`rpsA4` by default, also A3, A5, Letter, Legal and custom), an `Orientation` , `Margins` , a `Columns` count and a `Watermark` . A simple report has one page definition, which repeats as many times as the data needs.
- `Page.Bands` holds the bands. Each band has a `BandType` and a `Height` .
- `Band.Items` holds the items. An item has a `Kind` (`rikText` by default, also `rikLine` , `rikShape` , `rikImage` , `rikBarcode` , `rikRichText` and a few more), a position, a size and a `StyleName` .

The band type decides when a band prints:

BandType	Prints
`rbtReportTitle`	Once, at the start of the report, before the page header of the first page.
`rbtPageHeader`	At the top of every page.
`rbtColumnHeader`	Above the data, at the top of every page (of every column on a multi column page).
`rbtGroupHeader`	Once at the start of every group, see Groups and aggregates.
`rbtMasterData`	Once for every record of the data source named in `DataSourceName`.
`rbtDetailData`	Once for every detail record of the current master record.
`rbtGroupFooter`	Once at the end of every group, when its totals are complete.
`rbtReportSummary`	Once, after the last record.
`rbtPageFooter`	At the bottom of every page.
`rbtColumnFooter`, `rbtChild`	At the bottom of every column, and right after a parent band.

Bands have a few switches you will meet in the demos: `VisibleExpression` (print only when an expression is true), `StartNewPage`, `RepeatOnEveryPage` (for group headers) and `KeepTogether` (do not split the band across a page break). Items have a `VisibleExpression` too.

Units, positions and colours

- **Units.** `Report.Units` is `ruMillimeters` by default; `ruPoints` and `ruInches` are the alternatives. Every `Left`, `Top`, `Width`, `Height` and margin is entered in that unit. On an A4 page with 20 mm margins, the printable width is 210 − 2 × 20 = 170 mm, which is why the items of the demos are 170 wide.
- **Positions are relative to the band.** An item with `Top = 0` sits at the top of its band, wherever the band ends up on the page. Only the band heights and the order of the bands decide where things land.
- **Font sizes are always in points**, whatever `Units` says. The default font is Helvetica, 10 points, one of the standard PDF fonts.
- **Colours are `$RRGGBB`.** `TsgcReportColor` is platform neutral and stores red, green and blue in that order, which is the reverse of the Delphi `TColor` (`$BBGGRR`). `$1A3A5C` is the dark blue of the demos. The form file stores the colour as a decimal number, so `1719900` in the DFM is `$1A3A5C`.

Styles

`Report.Styles` holds named styles. A style has a `Font` (`Name`, `Size`, `Bold`, `Color` and more), a `BackColor`, a `Frame` (which sides, line width and colour), a `Padding`, and the alignments `HAlign` and `VAlign`. An item uses a style through its `StyleName`; an item without a style uses the defaults.

Styles keep a report consistent: the column captions of demo 02 all use `Header` (bold white text on dark blue), the cells use `Cell` (a thin grey line at the bottom). Change the style once and every item that uses it follows. A report has no sgcUICore theme; its look comes from its styles.

Expressions in square brackets

The `Text.Text` of a text item is a template: everything outside square brackets is printed as it is, and every `[ ... ]` is an expression evaluated when the report is prepared.

- **Fields** are written `Source.Field`, for example `[Products.Price]`, where `Source` is the `SourceName` of a data source.
- **Built in variables**: `Page` and `TotalPages` for page numbers, `Line` (the running number, from 1, of the record printed by the data band, handy for alternating rows), `RecNo`, `Date` and `Time`.
- **Operators**: `+ - * / div mod`, comparisons `= <> < <= > >=`, `and or not`, and `+` to join strings.
- **Functions**: `IIF`, `FormatFloat`, `FormatDateTime`, `Now`, `Upper`, `Lower`, `Trim`, `Copy`, `Pos`, `Replace`, `Round`, `Trunc`, `IsNull`, `Coalesce` and others, plus the aggregates `Sum`, `Count`, `Avg`, `Min` and `Max`.
- **Strings use single quotes**, as in Pascal: `[FormatDateTime('yyyy-mm-dd', Now())]`. Inside a Pascal string or a DFM string the quotes are doubled, which is why the form file of demo 01 shows `''yyyy-mm-dd''`.

`Text.DisplayFormat` formats the result with a `FormatFloat` or `FormatDateTime` picture, such as `'#,##0.00'` or `'dd mmm'`, but only when the whole text is one expression, such as `[Products.Price]`. When a text mixes words and an expression, put the format inside the brackets: `Average sale [FormatFloat('#,##0.00', Avg(Sales.Amount))]`.

Data sources

A report reads data through data source components. `TsgcReportDataSetSource` wraps any `TDataSet` (a `TClientDataSet`, a FireDAC query, an ADO query); `TsgcReportJSONSource` reads a JSON array of objects; `TsgcReportEventSource` asks your event handlers for each record.

Three names connect a source to the report:

1. The source's `SourceName`, for example `'Products'`. This is the name the report uses, not the component name.
2. The `DataSourceName` of a master data or detail data band, the same `'Products'`. The band prints once per record of that source.
3. The fields in the expressions, `[Products.Price]`.

And one call connects the source to the engine: `Engine.AddDataSource(srcProducts)`. There is no design time link between a source and an engine; the engine only knows the sources added to it in code, so every demo with data calls `AddDataSource` before preparing.

For master and detail data, a detail source names its master in `MasterSource`, and `MasterFields` and `DetailFields` list the fields that match (separated by semicolons). Before every detail band, the engine restarts the detail source and keeps only the records that match the current master record. Demo 05 uses this for invoices and their lines.

Groups and aggregates

A group header band has a `GroupExpression`, for example `'Sales.Region'`. The engine starts a new group whenever the value of that expression changes from one record to the next, and closes the previous group with its group footer. Because groups follow changes, **the records must arrive sorted by the group expression**; demo 03 sets `IndexFieldNames = 'Region;SaleDate'` on its dataset for that reason.

An aggregate accumulates over the scope of the band it is printed in. `[Sum(Sales.Amount)]` in a group footer is the total of the group and starts again at every group; in the report summary it is the total of the whole report; in a page footer it is the total of the page. `[Count()]` counts the records. A group footer sees the fields of the last record of its group, so `Total [Sales.Region]` prints the region that just ended.

Variables and events

A bare name in brackets, such as `[Company]`, is a variable. The engine answers it from three places, in this order: the built in variables, the `Name=Value` lines of `Engine.Variables` (set in the Object Inspector), and the `OnGetVariable` event, which your code handles for values computed at run time.

The engine also has an `OnBeforePrintBand` event with a `var aPrint: Boolean` parameter: set it to `False` to skip a band this time. Most conditions are simpler to write as a `VisibleExpression`, as demo 03 does.

Designed in the form, or built in code

The QuickStart demos keep the whole report in the form file: every page, band, item and style is set in the Object Inspector, and the code only fills the data and prepares. The larger demos in `Demos\Reports\Delphi` are older and mostly build the same collections in code, at run time. Both are the same objects, so you can mix them freely: design the layout in the form and change a text or add a band in code before preparing.

This is the heading of demo 01 built in code, together with a style and a variable answered by `OnGetVariable`:

```
procedure BuildReport(aReport: TsgcReport);
var
  oPage: TsgcReportPage;
  oBand: TsgcReportBand;
  oItem: TsgcReportItem;
begin
  aReport.Title := 'Hello sgcReports';
  with aReport.Styles.Add do
  begin
    Name := 'Heading';
    Font.Size := 20;
    Font.Bold := True;
    Font.Color := $1A3A5C; // $RRGGBB, not a TColor
  end;

  oPage := aReport.Pages.Add;
  oPage.Name := 'Page1';
  oPage.Margins.Left := 20;
  oPage.Margins.Top := 20;
  oPage.Margins.Right := 20;
  oPage.Margins.Bottom := 20;

  oBand := oPage.Bands.Add;
  oBand.Name := 'Title';
  oBand.BandType := rbtReportTitle;
  oBand.Height := 30;

  oItem := oBand.Items.Add;
  oItem.Kind := rikText;
  oItem.Width := 170;
  oItem.Height := 10;
  oItem.StyleName := 'Heading';
  oItem.Text.Text := 'Hello, sgcReports';

  oItem := oBand.Items.Add;
  oItem.Top := 14;
  oItem.Width := 170;
  oItem.Height := 6;
  oItem.Text.Text := 'Written by [Author] on [FormatDateTime(''yyyy-mm-dd'', Now())]';
end;
```

The units used are `sgcReports_Types`, `sgcReports_Model`, `sgcReports_Layout`, `sgcReports_Engine` and `sgcReports_Export_PDF`. For larger reports built in code there is also a fluent builder, `TsgcReportBuilder`, shown by the `ReportFluent` demo.

Prepared pages and threads

`Engine.Prepare` returns a `TsgcReportPreparedPages` object (unit `sgcReports_Layout`) with the finished pages. **The caller owns it** and must free it. The preview can show it with `Preview.LoadPrepared`, and every export component saves it with `SaveToFile`, so you prepare once and produce as many outputs as you like. Demo 04 works this way.

Nothing in this needs a window. The same report can be prepared and saved as a PDF in a console application or a service:

```
  oReport := TsgcReport.Create(nil);
  oEngine := TsgcReportEngine.Create(nil);
  oExport := TsgcReportPDFExport.Create(nil);
  try
    BuildReport(oReport);
    oEngine.Report := oReport;
    oEngine.OnGetVariable := oHelper.EngineGetVariable;
    oEngine.OnBeforePrintBand := oHelper.EngineBeforePrintBand;
    oPages := oEngine.Prepare;
    try
      oExport.SaveToFile(oPages, 'hello.pdf');
    finally
      oPages.Free;
    end;
  finally
    oExport.Free;
    oEngine.Free;
    oReport.Free;
  end;
```

Here `oHelper` is an object with the two event handlers, declared like this:

```
procedure THelper.EngineGetVariable(Sender: TObject; const aName: string;
  var aValue: Variant; var aHandled: Boolean);
begin
  if SameText(aName, 'Author') then
  begin
    aValue := 'The sales team';
    aHandled := True;
  end;
end;

procedure THelper.EngineBeforePrintBand(Sender: TObject; aBand: TsgcReportBand;
  var aPrint: Boolean);
begin
  // aPrint := False skips the band this time
  aPrint := aBand.Name <> 'Hidden';
end;
```

The threading rule is simple: **an engine, its report, its data sources and its exports belong to one thread at a time.** A server that prepares several reports at once gives every worker thread its own set of components. In a VCL application, prepare on the main thread, as the demos do, or prepare in a worker thread with its own engine and hand only the finished pages to the preview on the main thread.

4. The QuickStart demos

The five demos build on each other. Run them in order: each section below starts with a screenshot of the result, lists the components on the form, shows the important parts of the report layout and the code, explains what happens when the report is prepared, and ends with a few changes to try.

Folder	Project	New ideas
01.HelloReport	`HelloReport.dpr`	Report, engine and preview; pages, bands, items and styles; a first expression
02.ReportDataList	`ReportDataList.dpr`	A data source, the master data band, field expressions, `DisplayFormat`, styles with a background and a frame
03.ReportGroupTotals	`ReportGroupTotals.dpr`	Groups, aggregates, the summary band, conditional visibility, variables
04.ReportExportPrint	`ReportExportPrint.dpr`	Prepared pages, the export components, printing
05.ReportInvoiceApp	`ReportInvoiceApp.dpr`	Master and detail data, a new page per group, repeated headers, totals, filtering

> **Note.** The demos need sgcReports installed, so the components appear on the sgcReports palette page and the `sgcUI\Source` folder is on the library path. They use only features that are part of every edition. Demos 02 to 05 fill a `TClientDataSet` in code and use the `MidasLib` unit, so they need no `midas.dll`. All five call `Preview.FitWidth` after preparing, so the width of the page fills the window.

4.1 Demo 01: Hello Report

Folder `01.HelloReport`, project `HelloReport.dpr`.

What you will build

The smallest report there is. One A4 page with a title band (a heading, a subtitle, a horizontal line and three lines of text, the last one with the current date and time) and a page footer with the page number. There is no data: the point is to see the three components and the shape of a report definition.

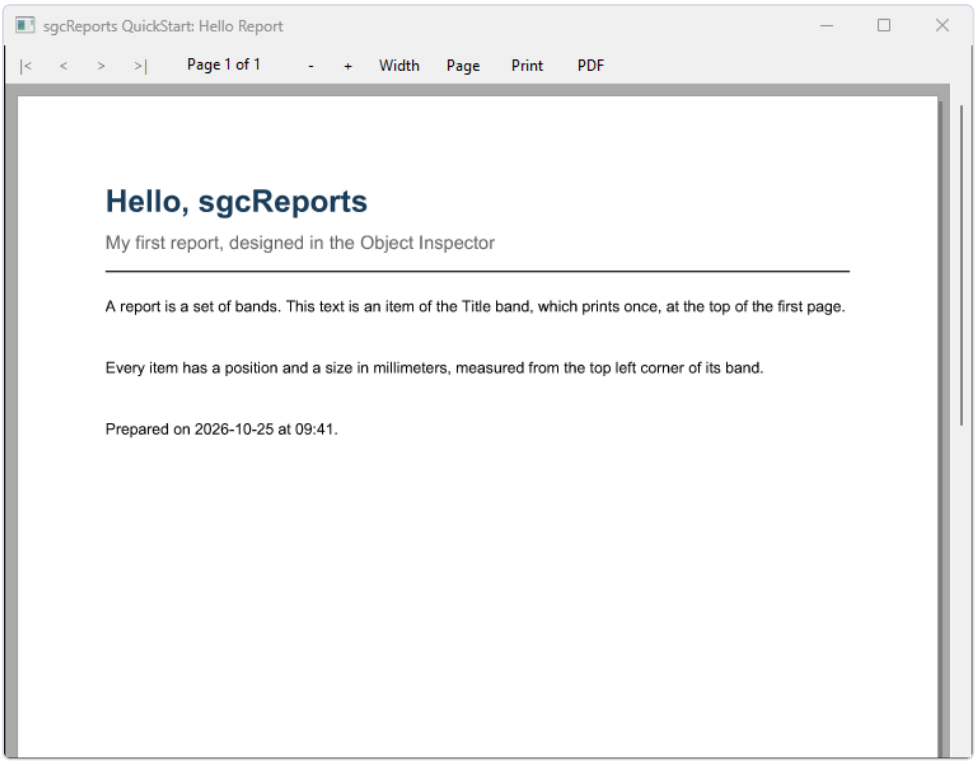


Figure 4.1. Demo 01. The preview shows the page with a toolbar to page, zoom, print and save a PDF.

Components on the form

Component	Class	Properties you set
`Report`	`TsgcReport`	`Title = 'Hello sgcReports'`. `Pages`: one page, `Page1`, with 20 mm margins and two bands, `Title` (`rbtReportTitle`, 80 mm high) and `Footer` (`rbtPageFooter`, 8 mm). `Styles`: `Heading`, `Subtitle` and `Small`.
`Engine`	`TsgcReportEngine`	`Report = Report`.
`Preview`	`TsgcReportPreviewVCL`	`Align = alClient`, `Engine = Engine`.

This is the title band of the page, with the first two items and the last one. The heading item has no `Kind` line because `rikText` is the default; the line item is a `rikLine` as wide as the page body:

```
      item
        Name = 'Title'
        BandType = rbtReportTitle
        Height = 80.000000000000000000
        Items = <
          item
            Width = 170.000000000000000000
            Height = 10.000000000000000000
            StyleName = 'Heading'
            Text.Text = 'Hello, sgcReports'
          end
          item
            Top = 11.000000000000000000
            Width = 170.000000000000000000
            Height = 6.000000000000000000
            StyleName = 'Subtitle'
            Text.Text = 'My first report, designed in the Object Inspector'
          end
          item
            Kind = rikLine
            Top = 20.000000000000000000
            Width = 170.000000000000000000
          end
          // ...
          item
            Top = 54.000000000000000000
            Width = 170.000000000000000000
            Height = 6.000000000000000000
            Text.Text =
              'Prepared on [FormatDateTime(''yyyy-mm-dd'', Now())] at [FormatD' +
              'ateTime(''hh:nn'', Now())].'
          end>
      end
```

And the three styles. The colours are `$RRGGBB` values stored as decimals: `1719900` is `$1A3A5C`, a dark blue, and `6579300` is `$646464`, a grey:

```
  Styles = <
    item
      Name = 'Heading'
      Font.Size = 20.000000000000000000
      Font.Bold = True
      Font.Color = 1719900
    end
    item
      Name = 'Subtitle'
      Font.Size = 12.000000000000000000
      Font.Color = 6579300
    end
    item
      Name = 'Small'
      Font.Size = 8.000000000000000000
      Font.Color = 6579300
      HAlign = rhaRight
    end>
```

The code

The `uses` clause names the units of the three components:

```
uses
  SysUtils, Classes, Controls, Forms,
  // sgc
  sgcReports_Model, sgcReports_Engine, sgcReports_Preview_VCL;
```

And this is the only event handler:

```
procedure TfrmHelloReport.FormCreate(Sender: TObject);
begin
  // lays out the report and shows the pages in the preview
  Preview.Prepare;
  // zoom so the width of the page fills the preview
  Preview.FitWidth;
end;
```

How it works

1. The form loads from its DFM. The report reads its page, bands, items and styles; the engine links to the report and the preview links to the engine.
2. `Preview.Prepare` asks the engine to prepare the report. The engine starts page 1, prints the title band at the top of the body and the page footer at the bottom, 20 mm from the edge of the paper.
3. While printing, the engine evaluates the expressions: `FormatDateTime('yyyy-mm-dd', Now())` becomes today's date, `[Page]` becomes 1. `[TotalPages]` is resolved in a second pass, once the engine knows how many pages there are.
4. The preview turns the prepared page into a PDF document and shows it. `FitWidth` sets the zoom.

Try this

- Set the page `Orientation` to `roLandscape`, or `PaperSize` to `rpsLetter`. The items keep their positions; widen them to use the extra space.
- Set `Report.Units` to `ruPoints` and run: every position is now read in points, so the layout shrinks to about a third.
- Set the page `Watermark.Kind` to `rwkText` and `Watermark.Text` to `DRAFT`.

4.2 Demo 02: Report Data List

Folder `02.ReportDataList`, project `ReportDataList.dpr`.

What you will build

A product list with twenty records from a `TClientDataSet`: a title, a row of column captions in white on dark blue, one line per product with code, name, category, price and stock, and the page footer of demo 01.

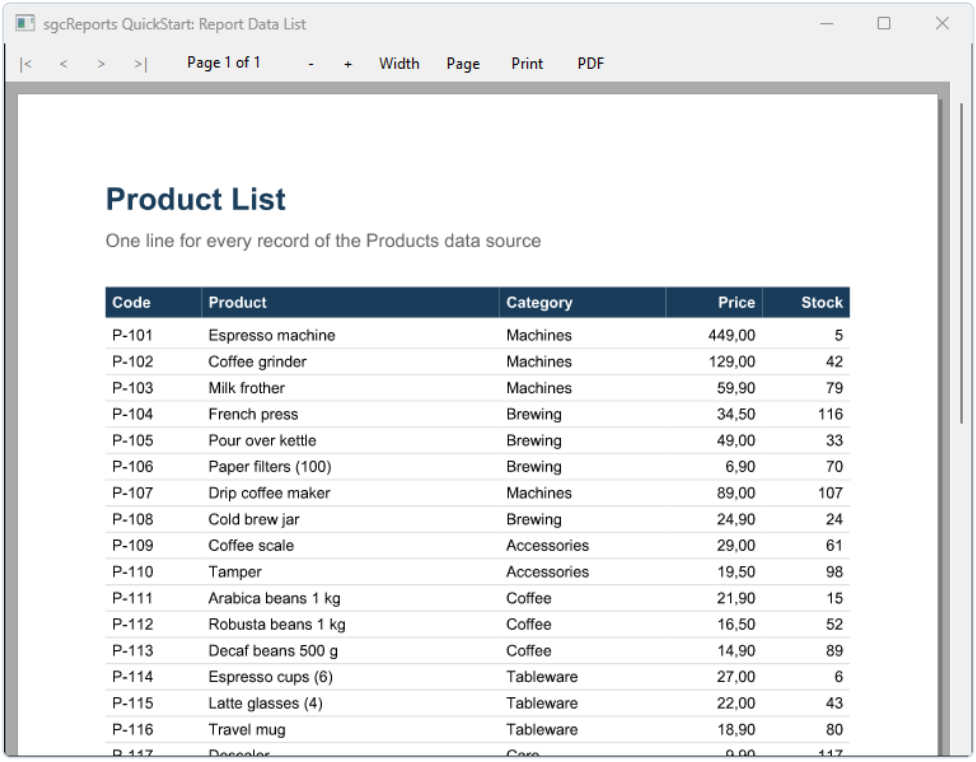


Figure 4.2. Demo 02. The master data band prints once for every record of the Products source.

Components on the form

Component	Class	Properties you set
`cdsProducts`	`TClientDataSet`	`FieldDefs`: `Code`, `Product`, `Category` (strings), `Price` (float), `Stock` (integer). `StoreDefs = True`.
`srcProducts`	`TsgcReportDataSetSource`	`SourceName = 'Products'`, `DataSet = cdsProducts`.
`Report`	`TsgcReport`	Bands `Title`, `ColumnHeader` (`rbtColumnHeader`), `ProductData` (`rbtMasterData`, `DataSourceName = 'Products'`) and `Footer`. Styles `Header`, `HeaderRight`, `Cell`, `Right` and others.
`Engine`, `Preview`		As in demo 01.

The data band names its source, and its items read the fields of the current record. Price and Stock have a `DisplayFormat`:

```
    item
      Name = 'ProductData'
      BandType = rbtMasterData
      Height = 6.00000000000000000
      DataSourceName = 'Products'
      Items = <
        item
          Width = 22.00000000000000000
          Height = 6.00000000000000000
          StyleName = 'Cell'
          Text.Text = '[Products.Code]'
        end
        // ...
        item
          Left = 128.00000000000000000
          Width = 22.00000000000000000
          Height = 6.00000000000000000
          StyleName = 'Right'
          Text.Text = '[Products.Price]'
          Text.DisplayFormat = '#,##0.00'
        end
        item
          Left = 150.00000000000000000
          Width = 20.00000000000000000
          Height = 6.00000000000000000
          StyleName = 'Right'
          Text.Text = '[Products.Stock]'
          Text.DisplayFormat = '#,##0'
      end>
    end
```

The column captions use a style with a background colour, white bold text and a little padding; the cells use a style with a thin light grey line at the bottom:

```
  item
    Name = 'Header'
    Font.Bold = True
    Font.Color = 16777215
    BackColor = 1719900
    Padding.Left = 1.50000000000000000
    VAlign = rvaCenter
  end
  // ...
  item
    Name = 'Cell'
    Frame.Bottom = True
    Frame.Width = 0.50000000000000000
    Frame.Color = 14737632
    Padding.Left = 1.50000000000000000
    VAlign = rvaCenter
  end
```

The code

```
procedure TfrmReportDataList.FormCreate(Sender: TObject);
begin
  DoLoadProducts;
  // the report finds the data by the SourceName of every source added here
  Engine.AddDataSource(srcProducts);
  Preview.Prepare;
  Preview.FitWidth;
end;


procedure TfrmReportDataList.DoLoadProducts;
var
  i: Integer;
begin
  { The field definitions are in the DFM, so CreateDataSet only has to
    create the empty table. The stock follows a fixed formula, so every
    run shows the same numbers. }
  cdsProducts.CreateDataSet;
  for i := Low(CS_PRODUCTS) to High(CS_PRODUCTS) do
    cdsProducts.AppendRecord([Format('P-%.3d', [101 + i]), CS_PRODUCTS[i],
      CS_CATEGORIES[i], CS_PRICES[i], 5 + (i * 37) mod 120]);
  cdsProducts.First;
end;
```

`CS_PRODUCTS`, `CS_CATEGORIES` and `CS_PRICES` are constant arrays at the top of the unit.

How it works

1. `DoLoadProducts` creates the in memory table from the field definitions of the DFM and appends twenty records.
2. `Engine.AddDataSource(srcProducts)` tells the engine that a source called `Products` exists. Without this call, preparing stops with `Data source "Products" not found`.
3. The engine prints the title, then the column header, then reaches the master data band. It moves the source to the first record and prints the band once per record, evaluating `[Products.Code]` and the others for each one.
4. `[Products.Price]` is the only expression of its text, so `DisplayFormat = '#,##0.00'` formats it, with the thousands and decimal separators of Windows.
5. When a page is full, the engine prints the page footer, starts a new page and prints the column header again before the next record.

> **Tip.** `TsgcReportDataSetSource` works with any `TDataSet`. Replace the client dataset with a FireDAC `TFDQuery` or an ADO query and keep `SourceName`: the report does not change.

Try this

- Set `cdsProducts.IndexFieldNames` to `Price`. The report follows the order of the dataset.
- Add an item to the data band with the text `[Products.Price * Products.Stock]` and `DisplayFormat = '#,##0.00'`, the value of the stock. Make room by narrowing the Product column.
- Set the page `Columns` to `2` and narrow the items so a line fits in half the width.

4.3 Demo 03: Report Group Totals

Folder `03.ReportGroupTotals`, project `ReportGroupTotals.dpr`.

What you will build

Twenty four sales of the first quarter of the current year, grouped by region. Every region starts with a heading and ends with a subtotal and the number of sales. Rows alternate between white and a light blue stripe, sales of 2,000 or more are printed in bold green, and a summary at the end shows the grand total, the average and the largest sale. The company name and the period in the title are variables. The report has two pages.

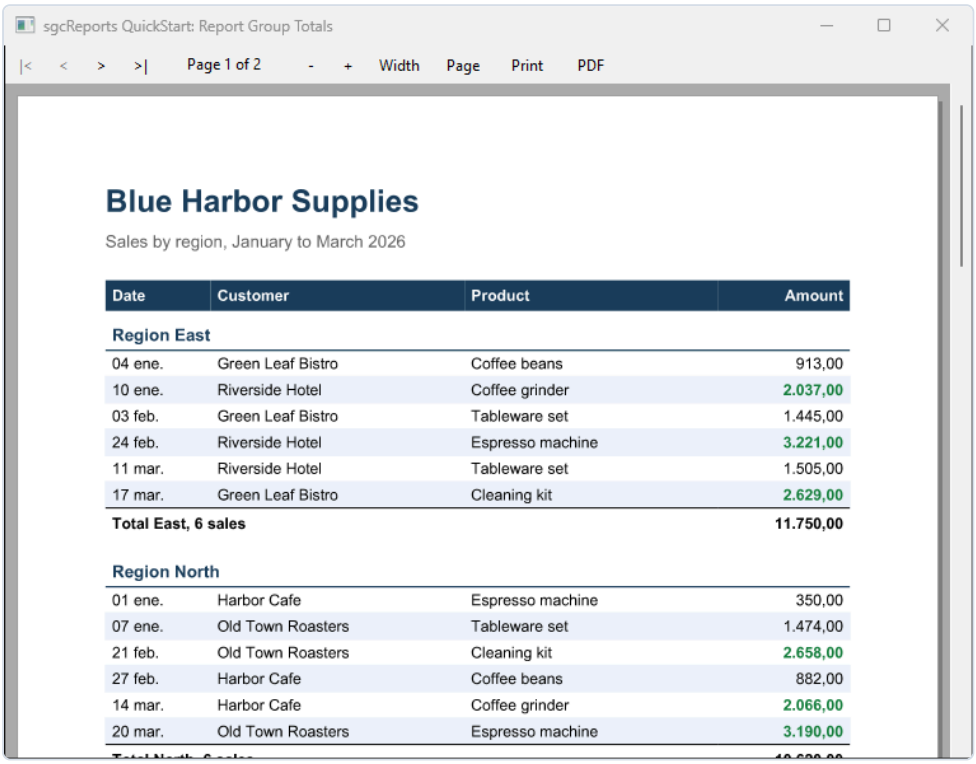


Figure 4.3. Demo 03. Each region is a group with a header and a footer; large amounts are bold green.

Components on the form

Component	Class	Properties you set
`cdsSales`	`TClientDataSet`	`FieldDefs`: `Region`, `SaleDate` (date), `Customer`, `Product`, `Amount` (float). `IndexFieldNames = 'Region;SaleDate'`.
`srcSales`	`TsgcReportDataSetSource`	`SourceName = 'Sales'`, `DataSet = cdsSales`.
`Report`	`TsgcReport`	Bands `Title`, `ColumnHeader`, `RegionHeader` (`rbtGroupHeader`), `SaleData` (`rbtMasterData`), `RegionFooter` (`rbtGroupFooter`), `Summary` (`rbtReportSummary`) and `Footer`.
`Engine`	`TsgcReportEngine`	`Variables`: `Company=Blue Harbor Supplies`. `OnGetVariable = EngineGetVariable`.

The group header, the data band and the group footer, in the order they appear in the `Bands` collection. The first item of the data band is a shape that draws the stripe on even lines; the amount appears twice in the same place, and the `VisibleExpression` of each copy decides which one prints:

```
        item
          Name = 'RegionHeader'
          BandType = rbtGroupHeader
          Height = 9.000000000000000000
          GroupExpression = 'Sales.Region'
          Items = <
            item
              Top = 2.000000000000000000
              Width = 170.00000000000000000
              Height = 7.000000000000000000
              StyleName = 'Group'
              Text.Text = 'Region [Sales.Region]'
            end>
        end
        item
          Name = 'SaleData'
          BandType = rbtMasterData
          Height = 6.000000000000000000
          DataSourceName = 'Sales'
          Items = <
            item
              Kind = rikShape
              VisibleExpression = 'Line mod 2 = 0'
              Width = 170.00000000000000000
              Height = 6.000000000000000000
              Shape.PenColor = 15660026
              Shape.FillColor = 15660026
            end
            item
              Width = 24.000000000000000000
              Height = 6.000000000000000000
              StyleName = 'Cell'
              Text.Text = '[Sales.SaleDate]'
              Text.DisplayFormat = 'dd mmm'
            end
            // ...
            item
              Left = 140.00000000000000000
              Width = 30.000000000000000000
              Height = 6.000000000000000000
              StyleName = 'Right'
              VisibleExpression = 'Sales.Amount < 2000'
              Text.Text = '[Sales.Amount]'
              Text.DisplayFormat = '#,##0.00'
            end
            item
              Left = 140.00000000000000000
              Width = 30.000000000000000000
              Height = 6.000000000000000000
              StyleName = 'Large'
              VisibleExpression = 'Sales.Amount >= 2000'
              Text.Text = '[Sales.Amount]'
              Text.DisplayFormat = '#,##0.00'
            end>
        end
        item
          Name = 'RegionFooter'
          BandType = rbtGroupFooter
          Height = 9.000000000000000000
          Items = <
            item
```

```
        Width = 140.00000000000000000
        Height = 7.00000000000000000
        StyleName = 'Total'
        Text.Text = 'Total [Sales.Region], [Count()] sales'
      end
      item
        Left = 140.00000000000000000
        Width = 30.00000000000000000
        Height = 7.00000000000000000
        StyleName = 'TotalRight'
        Text.Text = '[Sum(Sales.Amount)]'
        Text.DisplayFormat = '#,##0.00'
      end>
  end
```

The summary band uses the same aggregates over the whole report. Its last item mixes text and expressions, so the numbers are formatted with `FormatFloat` inside the brackets instead of `DisplayFormat`:

```
        Text.Text =
          'Average sale [FormatFloat(''#,##0.00'', Avg(Sales.Amount))], l' +
          'argest sale [FormatFloat(''#,##0.00'', Max(Sales.Amount))]'
```

The code

```
procedure TfrmReportGroupTotals.FormCreate(Sender: TObject);
begin
  DoLoadSales;
  Engine.AddDataSource(srcSales);
  Preview.Prepare;
  Preview.FitWidth;
end;


procedure TfrmReportGroupTotals.DoLoadSales;
var
  vMonth, vDay: Word;
  i, vRegion: Integer;
begin
  { Six sales per region in the first quarter of the current year. The
    amounts follow a fixed formula, so every run shows the same numbers.
    The records are added in any order: IndexFieldNames sorts them. }
  DecodeDate(Date, FYear, vMonth, vDay);
  cdsSales.CreateDataSet;
  for i := 0 to 23 do
  begin
    vRegion := i mod 4;
    cdsSales.AppendRecord([CS_REGIONS[vRegion],
      EncodeDate(FYear, 1 + i mod 3, 1 + (i * 5) mod 27),
      CS_CUSTOMERS[(i * 3) mod 8], CS_PRODUCTS[i mod 5],
      350 + ((i * 577) mod 2900) + (i mod 4) * 0.5]);
  end;
  cdsSales.First;
end;


procedure TfrmReportGroupTotals.EngineGetVariable(Sender: TObject;
  const aName: string; var aValue: Variant; var aHandled: Boolean);
begin
  { Called for every [Name] of the report that is not a field, a built in
    variable (Page, TotalPages, Line...) or an entry of Engine.Variables.
    Set aHandled to True when you answer it. }
  if SameText(aName, 'Period') then
  begin
    aValue := 'January to March ' + IntToStr(FYear);
    aHandled := True;
  end;
end;
```

How it works

1. The records are appended region after region in turn, but `IndexFieldNames = 'Region;SaleDate'` makes the dataset return them sorted by region and date. The groups depend on that order.
2. The title prints `[Company]` from `Engine.Variables` and `[Period]` from `EngineGetVariable`.
3. For the first record, the engine evaluates `Sales.Region`, opens a group and prints `RegionHeader`. It then prints `SaleData` for every record with the same region. `Line` counts the lines, so the stripe shape is visible on every second line, and only one of the two amount items is visible, depending on the amount.
4. When the region changes, the engine closes the group: it prints `RegionFooter`, where `Count()` and `Sum(Sales.Amount)` hold the values of that group only, and `[Sales.Region]` still shows the region that ended. Then it opens the next group.

5. After the last record it closes the last group and prints `Summary`, whose aggregates cover all twenty four sales.

Try this

- Add a nested group on `Sales.Customer`: a group header after `RegionHeader` and a group footer before `RegionFooter`, and change `IndexFieldNames` to `Region;Customer;SaleDate`.
- Set `StartNewPage = True` on `RegionHeader`, so every region starts on its own page.
- Add an item with the text `[IIF(Sales.Amount >= 2000, 'large', '')]` to the data band.

4.4 Demo 04: Report Export Print

Folder `04.ReportExportPrint`, project `ReportExportPrint.dpr`.

What you will build

The product list of demo 02 with a button bar above the preview: **Save PDF**, **Save Excel**, **Save CSV**, **Save PNG** and **Print...**. A label shows how many pages were prepared and, after each click, the file that was written. The files go next to the executable: `products.pdf`, `products.xlsx`, `products.csv` and one `products_page1.png` per page.

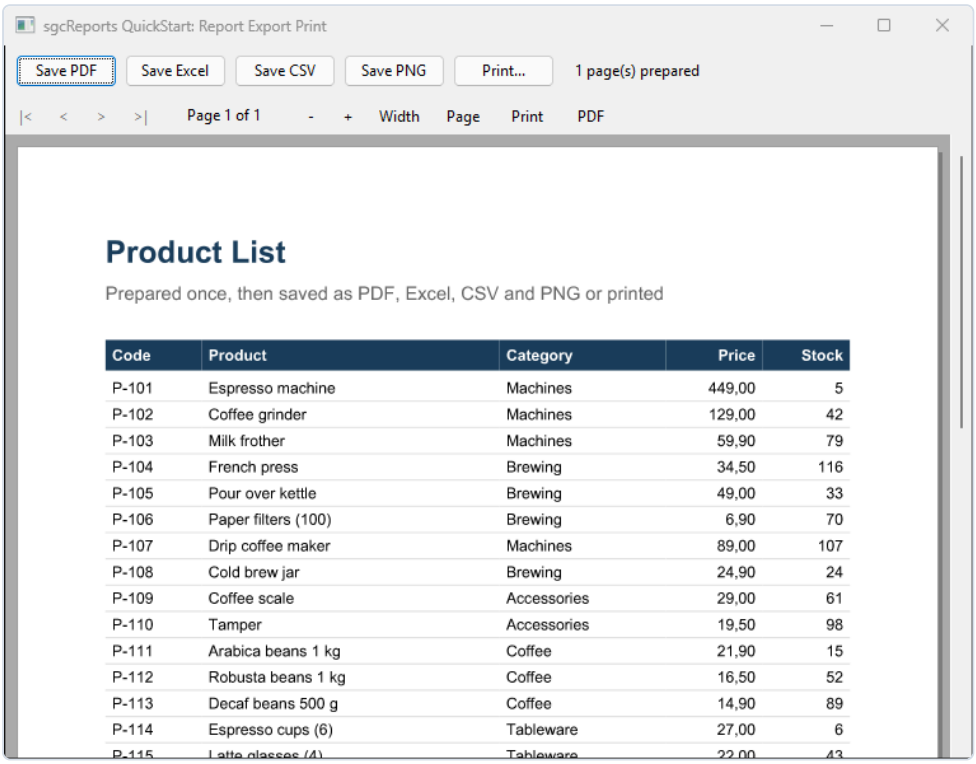


Figure 4.4. Demo 04. The pages are prepared once; the preview and the four export components share them.

Components on the form

Component	Class	Properties you set
`cdsProducts`, `srcProducts`, `Report`, `Engine`, `Preview`		As in demo 02.
`pdfExport`	`TsgcReportPDFExport`	`Title = 'Product List'`, `Author = 'Blue Harbor Supplies'`: the document properties of the PDF.
`xlsxExport`	`TsgcReportXLSXExport`	`SheetName = 'Products'`.
`csvExport`	`TsgcReportCSVExport`	`Delimiter = ';'`.
`imageExport`	`TsgcReportImageExport`	Defaults (`DPI = 150`).
`dlgPrint`	`TPrintDialog`	Selects the printer.
`pnlTop`	`TPanel`	Holds the five buttons and `lblStatus`.

The export components are configured entirely in the Object Inspector:

```
  object pdfExport: TsgcReportPDFExport
    Title = 'Product List'
    Author = 'Blue Harbor Supplies'
    Left = 200
    Top = 80
  end
  object xlsxExport: TsgcReportXLSXExport
    SheetName = 'Products'
    Left = 280
    Top = 80
  end
  object csvExport: TsgcReportCSVExport
    Delimiter = ';'
    Left = 360
    Top = 80
  end
  object imageExport: TsgcReportImageExport
    Left = 440
    Top = 80
  end
```

The code

The form keeps the prepared pages in a field, `FPages: TsgcReportPreparedPages`, declared in `sgcReports_Layout`. The `uses` clause adds that unit and the four export units:

```
uses
  SysUtils, Classes, Controls, Forms, StdCtrls, ExtCtrls, Dialogs, DB,
  DBClient, MidasLib,
  // sgc
  sgcReports_Model, sgcReports_Data, sgcReports_Layout, sgcReports_Engine,
  sgcReports_Export_PDF, sgcReports_Export_XLSX, sgcReports_Export_CSV,
  sgcReports_Export_Image, sgcReports_Preview_VCL;
```

```
procedure TfrmReportExportPrint.FormCreate(Sender: TObject);
begin
  DoLoadProducts;
  Engine.AddDataSource(srcProducts);
  // prepare the pages once: the preview and every export share them
  FPages := Engine.Prepare;
  Preview.LoadPrepared(FPages);
  Preview.FitWidth;
  lblStatus.Caption := Format('%d page(s) prepared', [FPages.Count]);
end;


procedure TfrmReportExportPrint.FormDestroy(Sender: TObject);
begin
  FPages.Free;
end;
```

Each button hands the same pages to a different export. `GetFileName` returns the folder of the executable plus the file name:

```
procedure TfrmReportExportPrint.btnPDFClick(Sender: TObject);
begin
  pdfExport.SaveToFile(FPages, GetFileName('products.pdf'));
  lblStatus.Caption := 'Saved ' + GetFileName('products.pdf');
```

```
end;


procedure TfrmReportExportPrint.btnXLSXClick(Sender: TObject);
begin
  // page header and page footer bands are left out of the sheet
  xlsxExport.SaveToFile(FPages, GetFileName('products.xlsx'));
  lblStatus.Caption := 'Saved ' + GetFileName('products.xlsx');
end;

// ...

procedure TfrmReportExportPrint.btnPNGClick(Sender: TObject);
begin
  // one file per page: %d is replaced by the page number
  imageExport.SaveToFiles(FPages, GetFileName('products_page%d.png'));
  lblStatus.Caption := Format('Saved %d PNG file(s), %s',
    [imageExport.PageCount(FPages), GetFileName('products_page1.png')]);
end;


procedure TfrmReportExportPrint.btnPrintClick(Sender: TObject);
begin
  // the dialog selects the printer; Print sends every page to it
  if dlgPrint.Execute then
  begin
    Preview.Print(Report.Title);
    lblStatus.Caption := 'Sent to the printer';
  end;
end;
```

How it works

1. `Engine.Prepare`, called directly this time, returns a new `TsgcReportPreparedPages`. The form owns it and frees it in `FormDestroy`.
2. `Preview.LoadPrepared` shows those pages without preparing again. (`Preview.Prepare`, used in the other demos, prepares into a temporary object and frees it after showing it.)
3. Every `SaveToFile` reads the same prepared pages. The report is not run again, so the four files always show exactly the same content as the preview.
4. The XLSX and CSV exports turn the text items into cells, using the item positions as columns. With `ExcludePageBands = True`, the default, they keep the page header and the column header only once and leave out the page footers, so the sheet is one continuous table. The image export draws one PNG per page at the resolution in `DPI`.
5. `Preview.Print` sends the pages to the current printer without a dialog. The demo shows a `TPrintDialog` first, so the user can choose the printer; the dialog makes its choice the current printer.

Try this

- Set `pdfExport.PDFA` to `rpaA3a` for an archival PDF/A-3a file. It stays accessible (`PDFUA = True` is the default).
- Set `imageExport.DPI` to `300` and compare the size of the PNG files.
- Set `Preview.ShowToolbar` to `False`: the button bar of the form is then the only way to print and save.

4.5 Demo 05: Report Invoice App

Folder `05.ReportInvoiceApp`, project `ReportInvoiceApp.dpr`.

What you will build

A small invoicing application. Three invoices for fictional customers, one per page, each with the company header, the invoice number, date and due date, a bill to block, a table of lines with alternating rows, the subtotal, the VAT and the total due, and a page footer. A combo box at the top shows all the invoices or one of them, and there are **Save PDF...** and **Print...** buttons. The dates are relative to today, so the demo never looks old.

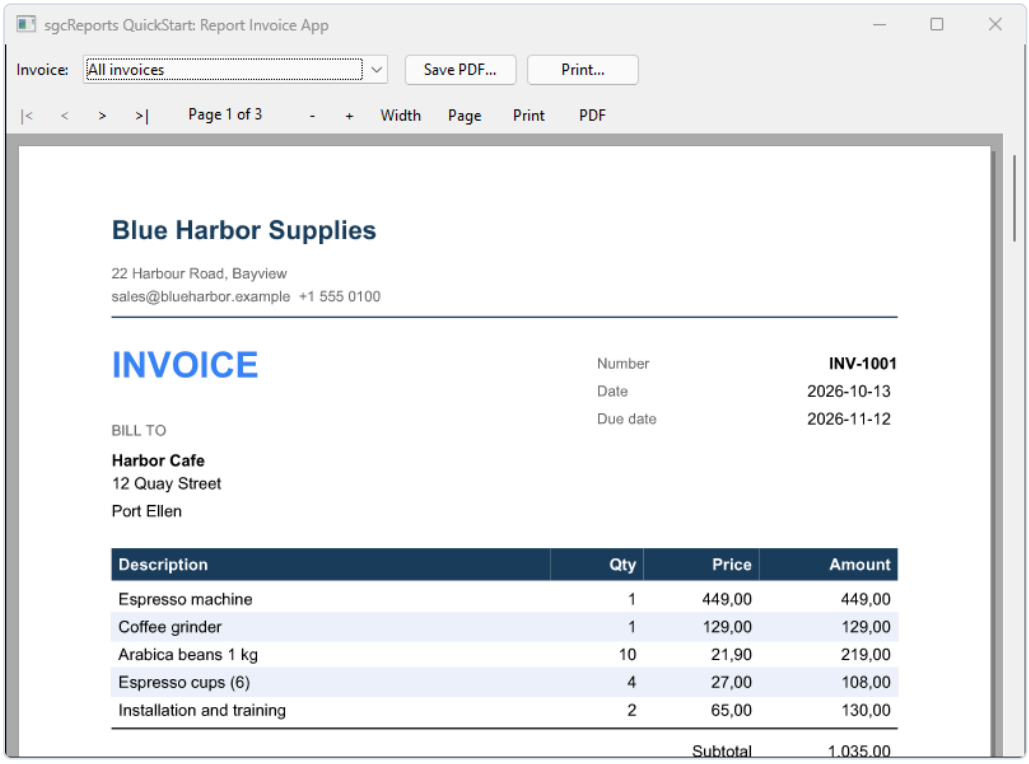


Figure 4.5. Demo 05. Each invoice is a group that starts a new page; its lines come from a detail source.

Components on the form

Component	Class	Properties you set
`cdsInvoices`	`TClientDataSet`	`Number`, `InvoiceDate`, `DueDate`, `Customer`, `Address`, `City`, `VATRate`.
`cdsLines`	`TClientDataSet`	`Invoice`, `Product`, `Qty`, `Price`.
`srcInvoices`	`TsgcReportDataSetSource`	`SourceName = 'Invoices'`, `DataSet = cdsInvoices`.
`srcLines`	`TsgcReportDataSetSource`	`SourceName = 'Lines'`, `DataSet = cdsLines`, `MasterSource = srcInvoices`, `MasterFields = 'Number'`, `DetailFields = 'Invoice'`.
`Report`	`TsgcReport`	Margins 20 mm left and right, 15 mm top and bottom. Bands `PageHeader`, `InvoiceHeader`, `LinesHeader`, `InvoiceData`, `LineData`, `Totals` and `PageFooter`.
`Engine`	`TsgcReportEngine`	`Variables`: `Company`, `CompanyAddress`, `CompanyContact`.
`cboInvoice`, `btnSavePDF`, `btnPrint`		The bar at the top.
`dlgSave`, `dlgPrint`	`TSaveDialog`, `TPrintDialog`	`dlgSave`: `DefaultExt = 'pdf'`, `FileName = 'invoices.pdf'`.

The bands, in the order of the `Bands` collection:

Band	BandType	What it does
`PageHeader`	`rbtPageHeader`	The company name, address and contact from `[Company]`, `[CompanyAddress]`, `[CompanyContact]`, and a line.
`InvoiceHeader`	`rbtGroupHeader`	`GroupExpression = 'Invoices.Number'`, `StartNewPage = True`: one invoice per page. The title, number, dates and bill to block.
`LinesHeader`	`rbtGroupHeader`	The same group, `RepeatOnEveryPage = True`: the column captions of the lines, repeated if an invoice spans pages.
`InvoiceData`	`rbtMasterData`	`DataSourceName = 'Invoices'`, no height and no items. It only drives the loop over the invoices.
`LineData`	`rbtDetailData`	`DataSourceName = 'Lines'`: one row per line of the current invoice, with a stripe and `[Lines.Qty * Lines.Price]`.
`Totals`	`rbtGroupFooter`	`KeepTogether = True`: subtotal, VAT and total due, never split across pages.
`PageFooter`	`rbtPageFooter`	A thank you line and `Page [Page] of [TotalPages]`.

The two group headers share the group expression; the first one starts a new page, the second one repeats:

```
        item
          Name = 'InvoiceHeader'
          BandType = rbtGroupHeader
          Height = 46.000000000000000000
          GroupExpression = 'Invoices.Number'
          StartNewPage = True
          Items = <
            // ...
        item
          Name = 'LinesHeader'
          BandType = rbtGroupHeader
          Height = 8.000000000000000000
          GroupExpression = 'Invoices.Number'
          RepeatOnEveryPage = True
          Items = <
            // ...
        item
          Name = 'InvoiceData'
          BandType = rbtMasterData
          DataSourceName = 'Invoices'
        end
```

The totals combine an aggregate over the detail lines with a field of the master record:

```
            item
              Left = 140.000000000000000000
              Top = 9.000000000000000000
              Width = 30.000000000000000000
              Height = 6.000000000000000000
              StyleName = 'Right'
              Text.Text = '[Sum(Lines.Qty * Lines.Price) * Invoices.VATRate / 100]'
              Text.DisplayFormat = '#,##0.00'
            end
            // ...
            item
              Left = 140.000000000000000000
              Top = 17.000000000000000000
              Width = 30.000000000000000000
              Height = 8.000000000000000000
              StyleName = 'GrandRight'
              Text.Text = '[Sum(Lines.Qty * Lines.Price) * (1 + Invoices.VATRate / 100)]'
              Text.DisplayFormat = '#,##0.00'
            end>
```

And the detail source names its master:

```
object srcLines: TsgcReportDataSetSource
  SourceName = 'Lines'
  MasterSource = srcInvoices
  MasterFields = 'Number'
  DetailFields = 'Invoice'
  DataSet = cdsLines
  Left = 120
  Top = 208
end
```

The code

```
procedure TfrmReportInvoiceApp.FormCreate(Sender: TObject);
begin
  DoLoadInvoices;
  // both sources are added: the report reads Invoices and Lines
  Engine.AddDataSource(srcInvoices);
  Engine.AddDataSource(srcLines);
  // the combo box: all the invoices first, then one entry per invoice
  cboInvoice.Items.Add('All invoices');
  cdsInvoices.First;
  while not cdsInvoices.Eof do
  begin
    cboInvoice.Items.Add(cdsInvoices.FieldByName('Number').AsString + '  ' +
      cdsInvoices.FieldByName('Customer').AsString);
    cdsInvoices.Next;
  end;
  cboInvoice.ItemIndex := 0;
  DoShowInvoices;
end;
```

`DoLoadInvoices` appends the three invoices and fourteen lines through two small helpers:

```
procedure TfrmReportInvoiceApp.DoLoadInvoices;
begin
  cdsInvoices.CreateDataSet;
  DoAddInvoice('INV-1001', 'Harbor Cafe', '12 Quay Street', 'Port Ellen',
    12, 21);
  // ...
  cdsLines.CreateDataSet;
  DoAddLine('INV-1001', 'Espresso machine', 1, 449.00);
  DoAddLine('INV-1001', 'Coffee grinder', 1, 129.00);
  // ...
end;
```

Choosing an invoice filters the master dataset and prepares again:

```
procedure TfrmReportInvoiceApp.DoShowInvoices;
var
  vNumber: string;
begin
  { Item 0 shows every invoice. Any other item filters the master dataset
    to one invoice: the report only sees the records the dataset shows. }
  if cboInvoice.ItemIndex <= 0 then
    cdsInvoices.Filtered := False
  else
  begin
    vNumber := Copy(cboInvoice.Text, 1, Pos(' ', cboInvoice.Text) - 1);
    cdsInvoices.Filter := 'Number = ' + QuotedStr(vNumber);
    cdsInvoices.Filtered := True;
  end;
  Preview.Prepare;
  Preview.FitWidth;
end;


procedure TfrmReportInvoiceApp.btnSavePDFClick(Sender: TObject);
begin
  // the preview already holds the PDF of the pages it shows
  if dlgSave.Execute then
    Preview.SaveToPDF(dlgSave.FileName);
end;
```

How it works

1. Both sources are added to the engine. Only `srcInvoices` drives a master data band; `srcLines` is read by the detail band and restarted for every invoice.
2. For each invoice record, the group expression `Invoices.Number` changes, so the engine opens a group: `InvoiceHeader` starts a new page (the page header prints at its top) and `LinesHeader` prints the column captions.
3. `InvoiceData` has no height, so the master band itself prints nothing. Under it, `LineData` prints the lines whose `Invoice` field equals the `Number` of the current invoice.
4. When the next invoice starts, or the data ends, the group closes and `Totals` prints. `Sum(Lines.Qty * Lines.Price)` covers only the lines of that invoice, and `Invoices.VATRate` is the rate of that invoice, 21 or 10 percent.
5. Filtering `cdsInvoices` hides the other invoices from the source, so `Preview.Prepare` produces only the selected invoice. `Preview.SaveToPDF` saves the document that the preview is showing.

Try this

- Add a QR code to `InvoiceHeader`: an item with `Kind = rikBarcode`, `Barcode.Symbology = rbsQRCode` and `Barcode.Value = '[Invoices.Number]'`, about 20 by 20 mm.
- Set the page `Watermark.Kind` to `rwkText` and `Watermark.Text` to `PAID`.
- Add forty more `DoAddLine` calls for `INV-1003` and watch `LinesHeader` repeat at the top of the second page of that invoice, while `Totals` stays in one piece.

5. Main components

The sgcReports palette page holds everything a first application needs: the report, the engine, a data source, the preview and the export components. This chapter introduces each of them, then the classes you meet inside the report, and finally the parts you will want later.

Every component has its own help topic, named after the class, such as `TsgcReportEngine.htm` or `TsgcReportPDFExport.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
`TsgcReport`	sgcReports	The report definition	`Pages`, `Styles`, `Title`, `Units`
`TsgcReportEngine`	sgcReports	Lays the report out into pages	`Report`, `Variables`, `AddDataSource`, `Prepare`, `OnGetVariable`, `OnBeforePrintBand`
`TsgcReportDataSetSource`	sgcReports	Records of a `TDataSet`	`SourceName`, `DataSet`, `MasterSource`, `MasterFields`, `DetailFields`
`TsgcReportJSONSource`	sgcReports	Records of a JSON array	`SourceName`, `JSON`, `FileName`, `RecordsPath`
`TsgcReportEventSource`	sgcReports	Records supplied by events	`SourceName`, `OnFirst`, `OnNext`, `OnEof`, `OnGetValue`
`TsgcReportPreviewVCL`	sgcReports	Shows, prints and saves the pages	`Engine`, `Prepare`, `LoadPrepared`, `FitWidth`, `Print`, `SaveToPDF`, `ShowToolbar`
`TsgcReportPDFExport`	sgcReports	PDF file	`SaveToFile`, `SaveToStream`, `Title`, `Author`, `PDFUA`, `PDFA`
`TsgcReportXLSXExport`	sgcReports	Excel workbook	`SaveToFile`, `SheetName`
`TsgcReportCSVExport`	sgcReports	CSV file	`SaveToFile`, `Delimiter`
`TsgcReportImageExport`	sgcReports	One PNG per page	`SaveToFiles`, `PageCount`, `DPI`
`TsgcVCLReportDesigner`	sgcReports	End user report designer (Registered editions)	See its help topic
`TsgcReportPage`, `TsgcReportBand`, `TsgcReportItem`, `TsgcReportStyle`	none, items of the collections	The parts of the definition	`Bands`, `BandType`, `Items`, `Kind`, `Text`, `StyleName`
`TsgcReportPreparedPages`	none, returned by `Prepare`	The finished pages	`Count`

The report definition

TsgcReport

A non visual component that only describes the report. `Pages` holds the page definitions with their bands and items, `Styles` the named styles, `Units` the unit of every position (millimetres by default) and `Title` a title that the preview uses when printing and that you can reuse elsewhere, as demo 04 does with `Preview.Print(Report.Title)`.

Besides the form file, a definition can be saved to and loaded from JSON with `sgcReportSaveToJSONFile` and `sgcReportLoadFromJSONFile` (unit `sgcReports_Definition`), which lets you store report layouts in a database or edit them outside Delphi. The `ReportJSONDefinition` and `ReportDatabaseSQL` demos use this.

Pages, bands, items and styles

You meet these classes as items of the collection editors. A **page** has the paper size, orientation, margins, columns and watermark. A **band** has the type, height, data source name, group expression and switches such as `StartNewPage`, `KeepTogether`, `RepeatOnEveryPage`, `CanGrow` and `VisibleExpression`. An **item** has the kind, position, size, style and the options of its kind: `Text` for texts, `Shape` for shapes, `Barcode` for barcodes, an image for pictures. A **style** has the font, colours, frame, padding and alignment.

The engine

TsgcReportEngine

The engine holds the run: the report it lays out (`Report`), the data sources added with `AddDataSource`, and the variables in `Variables` and `OnGetVariable`. `Prepare` returns a new `TsgcReportPreparedPages` that the caller frees; the overload `Prepare(aPages)` fills an object you already have. `RemoveDataSource` and `ClearDataSources` undo `AddDataSource`, which is useful when one engine runs different reports. Right click the engine at design time for **Preview...**.

The data sources

All three sources share `SourceName` and the master and detail properties `MasterSource`, `MasterFields` and `DetailFields`. They differ in where the records come from:

- **`TsgcReportDataSetSource`** reads any `TDataSet` through `DataSet`. Sorting and filtering are done on the dataset, as demos 03 and 05 show.
- **`TsgcReportJSONSource`** reads a JSON array of objects from `JSON`, from `FileName` or from a stream. `RecordsPath` is a dotted path to the array when it is not the root. The field names are taken from the first record. The `ReportGroups` demo reads `sales.json` this way.
- **`TsgcReportEventSource`** calls `OnFirst`, `OnNext`, `OnEof` and `OnGetValue`, for data that is neither a dataset nor JSON, such as a list of objects in memory or a paged web service.

The preview

TsgcReportPreviewVCL

A VCL control with a toolbar and a page view. Set `Engine` and call `Prepare`, or call `LoadPrepared` with pages you prepared yourself. The methods `FirstPage`, `PriorPage`, `NextPage`, `LastPage`, `ZoomIn`, `ZoomOut`, `FitWidth` and `FitPage` do what the toolbar buttons do, `PageCount` tells how many pages there are, and `OnPageChange` fires when the user pages. `Print` prints to the current printer and `SaveToPDF` saves the document being shown. `ShowToolbar` hides the toolbar when your form has its own buttons. The FireMonkey twin is `TsgcReportPreviewFMX`.

The export components

Every export takes a `TsgcReportPreparedPages`, so one preparation can feed several of them:

- **`TsgcReportPDFExport`**: `SaveToFile` and `SaveToStream`. `PDFUA` (`True` by default) writes a tagged, accessible PDF; `PDFA` chooses an archival level (`rpaNone`, `rpaA3b`, `rpaA3a`, `rpaA4`); `Title` and `Author` fill the document properties.
- **`TsgcReportXLSXExport`** and **`TsgcReportCSVExport`**: one continuous table built from the text items. `SheetName` names the worksheet, `Delimiter` separates the CSV fields.
- **`TsgcReportImageExport`**: `SaveToFiles` with a file name pattern containing `%d` writes one PNG per page at `DPI` dots per inch.

Beyond the basics

When the QuickStart feels familiar, these are the parts to look at next. Each has a help topic and most have a demo:

- **The fluent builder**, `TsgcReportBuilder` (unit `sgcReports_Fluent`), builds a whole report in a few chained calls. See the `ReportFluent` and `ReportMobileFMX` demos.
- **More item kinds**: barcodes and QR codes, images, rich text, subreports and crosstabs; watermarks on the page.
- **Report scripting**: `TsgcReport.Script` and the script properties of bands and items, for logic that travels with the definition.
- **The HTML export**, `TsgcReportHTMLExport`, created in code.
- **Accessible and archival PDF**: PDF/UA tagging, alternate texts on items, PDF/A. See the `ReportPDFUA` and `ReportInvoice` demos.
- **Registered editions**: the end user designer (`TsgcVCLReportDesigner`, `TsgcFMXReportDesigner`), the DOCX export, chart and pivot table items from sgcCharts and sgcPivotGrid, and sgcReportsHTML with `TsgcReportHTTPEndpoint` and a browser viewer, which also needs sgcWebSockets.

6. Common problems

These are the problems that come up most often in a first report. Each entry says what you see, why it happens and how to fix it. Most of them show up as an exception when the report is prepared, so when the form does not appear, read the message: it names the expression or the data source that failed.

Expressions

An exception with "Unexpected" when the report is prepared

What you see. An exception such as `Unexpected "yyyy-mm-dd" at position 29`, shown before the form appears.

Why. A string literal inside an expression was written with double quotes, as in `[FormatDateTime("yyyy-mm-dd", Now())]`. The expression language uses single quotes, like Pascal, and does not accept double quotes.

Fix. Use single quotes: `[FormatDateTime('yyyy-mm-dd', Now())]`. In the Object Inspector you type them once; in Pascal code and in the form file they are doubled, `''yyyy-mm-dd''`.

The number format is ignored

What you see. An item with `Text.DisplayFormat = '#,##0.00'` prints `449` instead of `449.00`.

Why. `DisplayFormat` applies only when the whole text is a single expression, such as `[Products.Price]`. A text like `Price: [Products.Price]` mixes literal text and an expression, and is printed without the format.

Fix. Format inside the brackets: `Price: [FormatFloat('#,##0.00', Products.Price)]`. For dates use `FormatDateTime`. Or split the text into a caption item and a value item.

A variable prints nothing

What you see. `[Company]` prints an empty text.

Why. A name that is not a field, a built in variable, a line of `Engine.Variables` or a name answered by `OnGetVariable` evaluates to Null, which prints as nothing. A missing field does the same. Typical causes are a typo, a `Variables` line without the `=`, or an `OnGetVariable` handler that forgets `aHandled := True`.

Fix. Check the spelling (case does not matter), the `Name=Value` format of `Variables`, and set `aHandled` in the event. Use `Coalesce` or `IsNull` in the expression when a value may really be missing.

Data and layout

Data source "Products" not found

What you see. Preparing raises `Data source "Products" not found`, at run time or in the design time **Preview...** window.

Why. A band has `DataSourceName = 'Products'`, but the engine has no source with that `SourceName`. A data source component on the form is not linked to the engine automatically, and there is no property to link it at design time: the engine only knows the sources passed to `AddDataSource`.

Fix. Call `Engine.AddDataSource(srcProducts)` before `Prepare`, for every source, detail sources included. Check that the `SourceName` of the source and the `DataSourceName` of the band are the same text. At design time, preview only reports without data bands.

A group appears several times

What you see. The same region has two or three headers and subtotals, scattered through the report.

Why. A group starts whenever the value of its `GroupExpression` changes from one record to the next. If the records are not sorted by that value, the value changes back and forth and the group opens again.

Fix. Sort the data by the group expressions, outer group first: `IndexFieldNames` on a client dataset, `ORDER BY` in a query.

The colours are wrong

What you see. A dark blue comes out as a brown, or red comes out blue.

Why. Report colours are `TsgcReportColor`, written `$RRGGBB`. A Delphi `TColor` is `$BBGGRR`, so a value copied from a `TColor` constant has red and blue swapped.

Fix. Write report colours as `$RRGGBB`, the same way as in HTML: `$1A3A5C` is `#1A3A5C`. Remember that the form file shows them as decimals.

Items are in the wrong place or far too small

What you see. The layout is squeezed into the top left corner, or items run off the page.

Why. Positions and sizes are in `Report.Units`, millimetres by default, and they are measured from the top left corner of the band, not of the page. Values meant for points in a millimetre report (or the other way round) are almost three times too big or too small.

Fix. Check `Report.Units` and keep every item inside the width of the page body: paper width minus the left and right margins, 170 mm on A4 with 20 mm margins. Font sizes are always in points.

The detail lines do not print

What you see. The invoices print their headers and totals, but no lines.

Why. A `rbtDetailData` band prints only when its source names the source of the master data band in `MasterSource`. With `MasterSource` empty, or pointing to another source, the engine skips the detail band.

Fix. Set `MasterSource` of the detail source to the master source and list the matching fields in `MasterFields` and `DetailFields`, the same number on both sides, separated by semicolons (a different number raises `MasterFields and DetailFields of "Lines" must list the same number of fields`). Remember to add the detail source with `AddDataSource` too.

Output, editions and compiling

Print does not ask for a printer

What you see. `Preview.Print` starts printing straight away on the default printer.

Why. `Print` sends every page to the current printer and shows no dialog.

Fix. Show a `TPrintDialog` first, as demos 04 and 05 do. The printer the user chooses becomes the current printer.

A memory leak report on exit

What you see. With `ReportMemoryLeaksOnShutdown` on, Delphi reports a leaked `TsgcReportPreparedPages` and the objects it holds.

Why. `Engine.Prepare` returns a new object that the caller owns, and nobody freed it.

Fix. Keep the pages in a field and free them in `FormDestroy`, as demo 04 does, and free the old pages before you keep the result of a new `Prepare`. Or call `Preview.Prepare`, which frees its pages itself.

The designer or the DOCX export is missing

What you see. `TsgcVCLReportDesigner` is not on the palette, the engine has no **Design report...** command, or `sgcReports_Export_DOCX` does not compile into your project.

Why. You are using the Community edition, which leaves out the advanced features (the designer, the DOCX export, the chart, pivot table and web bridges).

Fix. Use the Registered edition, or stay with the features of every edition, as the QuickStart demos do.

Units not found when compiling

What you see. `File not found: 'sgcReports_Model.dcu'`, or the same for an sgcPDF unit.

Why. The `sgcUI\Source` folder (or the folder of the compiled units) is not on the library path of this Delphi version and platform, typically after adding a Win64 target or installing a new Delphi version.

Fix. Run the sgcUI installer for that Delphi version, or add the folder in **Tools** > **Options** > **Language** > **Delphi Options** > **Library** for the platform you build. sgcReports needs sgcPDF, which lives in the same folder.

7. Where next

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

The other Delphi demos

The other folders in `Demos\Reports\Delphi` are complete applications. Most of them build their report in code, the style described in Designed in the form, or built in code. Each one has a `readme.html` that explains what it shows and how to run it.

Demo	What it shows
`ReportExports`	One report prepared once and exported to PDF, XLSX, CSV and one PNG per page. The natural next step after demo 04.
`ReportGroups`	A sales list read from `sales.json`, grouped by region and customer, with repeated group headers, group footers, page sums and a grand total.
`ReportInvoice`	Master and detail invoices from two client datasets, previewed and saved as PDF/A-3a that is also PDF/UA-1.
`ReportFluent`	A sales report grouped by region, built entirely in code with `TsgcReportBuilder`, and exported to PDF, XLSX, CSV and PNG.
`ReportJSONDefinition`	Edit a report definition and its data as JSON in two memos, render it, and save the current report back to JSON.
`ReportDatabaseSQL`	Reports configured from a database: a SQLite catalog with the SQL, the parameters and the JSON definition of every report, run through FireDAC.
`ReportPDFUA`	An accessible report with headings, tables with header rows, an image with alternate text and a barcode, exported as PDF/UA-1 and PDF/A.
`MigrateFromQuickReport`	A typical QuickReport master and detail invoice list, grouped by customer, rebuilt with sgcReports; every band and item names the QuickReport class it replaces.
`ReportLinuxDaemon`	A console report daemon for Windows and Linux64 that watches a jobs folder and renders each JSON job on a pool of worker threads.
`ReportMobileFMX`	A small A5 delivery note built with the fluent API in a FireMonkey preview, for Windows, macOS, iOS and Android.
`ReportCrosstabChart`	A sales report with a crosstab, a chart item and a pivot table item, exported to PDF, DOCX and HTML.
`ReportDesigner`, `ReportDesignerFMX`	The end user report designer for VCL and for FireMonkey, with a sample invoice that has rich text, a DRAFT watermark and a crosstab.
`ReportHTTPServer`	A report served over HTTP as PDF, XLSX, CSV, PNG and JSON with `TsgcReportHTTPEndpoint`, plus a page with the browser viewer.

`ReportCrosstabChart`, `ReportDesigner`, `ReportDesignerFMX` and `ReportHTTPServer` use features of the Registered editions, and `ReportHTTPServer` also needs sgcWebSockets. Their readme files say what they need.

Help topics

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

Topic	What it covers
Introduction	What sgcReports does and where to start.
Installation, Editions	Requirements, the installer, what each edition includes.
Report model	Pages, bands, items and styles in depth, with every band type and item kind.
Expressions	Fields, variables, operators, every function and the aggregates.
Data sources, Groups, headers, footers and columns	The three source kinds, master and detail, groups, keep together, columns and subreports.
Grouped report totals, Invoice with master and detail, Export to XLSX, CSV and PNG	Step by step versions of demos 03, 05 and 04.
PDF/A, Accessibility	Archival and accessible PDF output.
Thread safety and server use	Preparing reports on many threads at once.
One topic per component	`TsgcReport`, `TsgcReportEngine`, `TsgcReportDataSetSource`, `TsgcReportPreviewVCL`, `TsgcReportPDFExport` and the others, with every property, its type and its default.

On the website

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

Enjoy turning your data into reports.

---

sgcReports, sgcPDF and sgcUI are products of eSeGeCe. https://www.esegece.com