sgcMaps

Getting Started

Put an interactive map with markers, shapes and events 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 show a map in it with sgcMaps. You do not need to know anything about maps, tiles or map projections in advance. Chapter 1 explains the few ideas you need in a minute, and the rest of the guide builds on them step by step.

The guide follows the five QuickStart demos. Each demo adds one idea to the previous one, from a map with two markers placed in the Object Inspector to a small store locator with a list, a search box, a delivery area, a dark theme, an optional OpenStreetMap layer and an image export. 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 and zoom the map, click the markers, then read its section and try the suggested changes. Chapters 1 to 3 explain what sgcMaps 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\Maps\00.QuickStart` folder of your sgcUI installation. Each demo has its own subfolder, from `01.HelloMap` to `05.MapStoreApp`, with a Delphi project (`.dpr`) that you can open and run directly. All of them are VCL applications for Windows, and none of them needs an internet connection.

The screenshots were taken on a Windows machine with Spanish regional settings. Numbers in the demos, such as the coordinates in the log of demo 04, follow the Windows regional settings of the machine that runs the demo, so on your machine the decimal separator may differ from the one in the figures.

Contents

1. What sgcMaps is

The idea in one minute
Your first map needs one line of code
What you need
Part of sgcUI
Coming from browser based maps

2. Install and the palette

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

3. Core concepts

Coordinates, centre and zoom
The base map comes from layers
About the world outline in the demos
Overlays: the item collections
Markers
Shapes and labels
Events
Look and behaviour
Images, printing and threads
Summary: who does what

4. The QuickStart demos

4.1 Demo 01: Hello Map
4.2 Demo 02: Map Markers
4.3 Demo 03: Map Shapes
4.4 Demo 04: Map Events
4.5 Demo 05: Store Locator

5. Main components

Reference table
The map control
Layers and items
The online services
Offline data and location

6. Common problems

The base map
Markers, labels and colours
Versions, editions and output

7. Where next

The other Delphi demos
Help topics
On the website
Getting help

1. What sgcMaps is

sgcMaps adds interactive maps to Delphi applications, for the VCL and for FireMonkey. You drop a map control on a form, tell it where to look and how close, and the user can drag it with the mouse, zoom with the wheel or the zoom buttons, and click the things you placed on it: markers, labels, circles, rectangles, lines and areas.

The map is a native control. It is painted by Delphi code, not by a browser, and it reads its map images from wherever you point it: a public tile server such as OpenStreetMap, a commercial provider with an API key, your own server, or files on disk that work with no network at all. Next to the control, a set of non visual components looks up addresses, calculates routes, finds places and reads the elevation or the time zone of a point.

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

A map in sgcMaps is two things laid on top of each other:

- **The base map.** The picture of the world: land, sea, roads, names. It is made of *tiles*, small square images that cover the world at every zoom level. At zoom 0 one tile shows the whole world, and every further level doubles the detail. The `Layers` property of the map says where the tiles come from.
- **The overlays.** Everything your application adds: markers with a popup and a tooltip, text labels, circles, rectangles, polylines and polygons. Each kind has its own collection on the map, such as `Markers` or `Circles`, and every item has a position in degrees of latitude and longitude.

Three properties decide what the user sees when the form opens: `CenterLatitude` and `CenterLongitude`, the point in the middle of the map, and `Zoom`, how close the map is. Everything else is detail.

> **Important.** sgcMaps has no built-in world data. A map with an empty `Layers` collection, which is how a new map starts, paints only its background colour and your overlays. A real application adds a tile layer. The QuickStart demos must run without internet, so they draw a simple outline of the continents instead, as explained in About the world outline in the demos.

Your first map needs one line of code

The first QuickStart demo is the smallest map there is. Everything is set in the Object Inspector, and the form file stores it like this:

```
object Map: TsgcVCLMap
  Align = alClient
  CenterLatitude = 46.000000000000000000
  CenterLongitude = 12.000000000000000000
  Zoom = 3
  Markers = <
    item
      Latitude = 40.416800000000000000
      Longitude = -3.703800000000000000
      PopupText = 'Madrid, capital of Spain'
      TooltipText = 'Madrid'
    end
    item
      Latitude = 48.856600000000000000
      Longitude = 2.352200000000000000
      PopupText = 'Paris, capital of France'
      TooltipText = 'Paris'
    end>
  Options.ShowAttribution = False
  Options.BackgroundColor = 15259050
end
```

The only code adds the land, with a helper unit shared by the QuickStart demos:

```
procedure TfrmHelloMap.FormCreate(Sender: TObject);
begin
  { Adds the continents as polygons, see uQSWorldMap.pas. }
  AddWorldOutline(Map);
end;
```

That is all. Run it and you get a map of Europe with two red pins. Drag it, zoom it with the wheel or the + and − buttons, hover a pin to see its tooltip and click it to open its popup. The zoom buttons, the scale bar, the dragging and the popups are built in; you did not write any code for them.

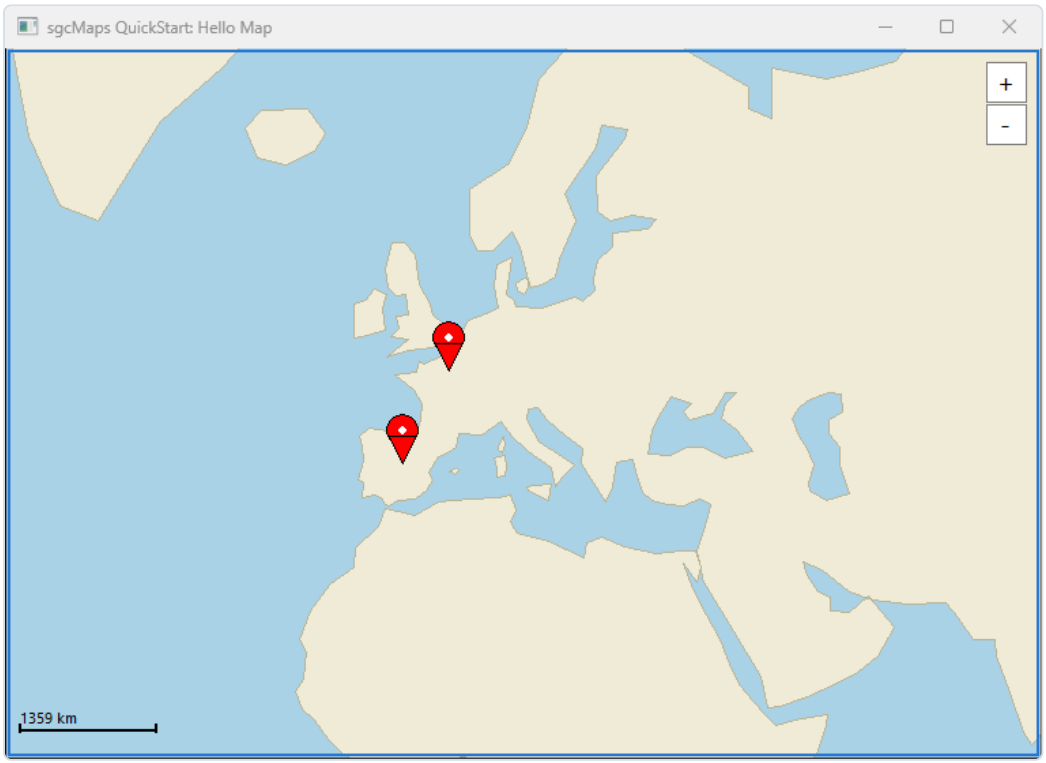


Figure 1.1. The result of demo 01: a map centred on Europe at zoom 3 with two markers, the zoom buttons and the scale bar.

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.** `TsgcVCLMap` runs on Windows, Win32 in every version and Win64 where your Delphi version can build it.
- **FireMonkey.** `TsgcFMXMap` has its own packages from Delphi XE6 on. It runs on Windows from XE6, and on macOS, iOS and Android from Delphi 10.4.
- **sgcUICore.** The shared foundation of the sgcUI visual products. It is installed together with sgcMaps, you do not install it separately.

Some features need a newer Delphi than the map itself. The map control and the overlays work from Delphi 7, with tiles read from a folder. The table below lists the rest:

Feature	From
Tiles from a server (providers, URL templates, WMS, WMTS)	Delphi 2009
`SaveToPNG`	Delphi 2009 (`SaveToBMP` works in every version)
Vector tiles and PMTiles archives	Delphi XE2
`TsgcMapRouteCalculator` , `TsgcMapOfflineRouter`	Delphi XE2
`TsgcMapLocation` , the FireMonkey map	Delphi XE6
MBTiles files, `TsgcMapOfflineGeocoder`	Delphi XE7
The online services: geocoder, directions, places, elevation, time zone	Delphi XE8

The QuickStart demos use only what Delphi 7 has, so they compile with every supported version. Demo 05 has an optional OpenStreetMap layer that shows tiles only from Delphi 2009 on.

Part of sgcUI

sgcMaps is one product of sgcUI, the eSeGeCe suite of visual components (grids, charts, dashboards, planners and more). The products share sgcUICore, the layer with the light and dark themes, the printing and the page settings. In practice this means two things for you:

- The `Theme` property (`utDefault` , `utLight` , `utDark` , `utCustom` , `utHighContrast`) works the same way on the map as on the other sgcUI controls on your form. On the map it paints the controls drawn over the map: the zoom buttons, the scale bar, the popups and the layer switcher.
- Printing and the PDF export of the map (`Print` , `ExportToPDF` , `PrintSettings`) go through sgcUICore, like those of the other sgcUI products.

When sgcDashboard is installed as well, the installer adds bridge packages that connect the two products.

Coming from browser based maps

A common way to show a map in a Delphi application is to host a web page with a JavaScript map library in a browser control, and to pass data between Delphi and the page as script calls. sgcMaps takes a different route: the map is an ordinary Delphi control. Markers and shapes are collection items you edit in the Object Inspector

or in Pascal, events are ordinary Delphi events with typed parameters, and there is no browser runtime to deploy and no web page to keep in step with your code.

Because the control decides where its tiles come from, the same application can show OpenStreetMap while online and a local MBTiles or PMTiles file while offline, with the same markers and the same code. Chapter 7 points to the demos that do this.

2. Install and the palette

This chapter gets sgcMaps 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

sgcMaps 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 **sgcMaps**. The installer shows one checkbox per product and enables the products your licence includes. sgcUICore has no checkbox: it is installed automatically with sgcMaps.
4. The installer copies the files, compiles the packages on your machine for each selected Delphi version and installs the design time packages.

Everything goes into one folder, by default `Documents\esegece\sgcUI`. The installer adds its `Source` folder and the folder of the compiled units to the library path of each selected Delphi version, so a new project finds the sgcMaps units without any setting on your side. With sgcMaps the installer also copies the `Tools\Maps` folder, with the tools that build the data files of the offline geocoder and router.

The packages

You only need this section if you build the packages yourself. sgcMaps 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
`sgcMaps<V>`	Runtime	The map model, the items, the tile sources, the file formats, the geometry functions, the services and the offline data. No VCL, no FireMonkey.
`sgcMapsVCL<V>`	Runtime	`TsgcVCLMap`.
`sgcMapsFMX<V>`	Runtime, XE6 and later	`TsgcFMXMap`.
`dclsgcMaps<V>`	Design time	Registers the VCL map and the non visual components on the palette.
`dclsgcMapsFMX<V>`	Design time, XE6 and later	Registers the FireMonkey map.

Build them in dependency order: sgcUICore first, then `sgcMaps`, then `sgcMapsVCL`, then `dclsgcMaps`. 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 Maps**. In a VCL project it shows these components, depending on your Delphi version and edition:

Component	Visual	What it does
`TsgcVCLMap`	yes	The map. Every QuickStart demo uses only this component.
`TsgcMapGeocoder`	no	Turns an address into coordinates and coordinates into an address. Delphi XE8 and later.
`TsgcMapDirections`	no	Asks an online routing service for a route between points. Delphi XE8 and later.
`TsgcMapPlaces`	no	Searches places and gives suggestions while the user types. Delphi XE8 and later.
`TsgcMapElevation`	no	Returns the elevation of points. Delphi XE8 and later.
`TsgcMapTimeZone`	no	Returns the time zone of a point. Delphi XE8 and later.
`TsgcMapRouteCalculator`	no	Calculates routes. Delphi XE2 and later.
`TsgcMapOfflineRouter`	no	Calculates routes from a local graph file, with no network. Delphi XE2 and later, Registered and Trial editions.
`TsgcMapOfflineGeocoder`, `TsgcMapGeoIndexBuilder`	no	Searches places in a local database, and builds that database. Delphi XE7 and later, Registered and Trial editions.
`TsgcMapLocation`	no	The position of the device. Delphi XE6 and later.

In a FireMonkey project the same page shows `TsgcFMXMap` instead of the VCL map, together with the same non visual components, which do not depend on a framework.

> **Tip.** To edit the markers or the shapes at design time, select the map, find `Markers` (or `Circles`, `Rectangles`, `Labels`) in the Object Inspector and click its ellipsis button. The collection editor lists one item per marker or shape; select an item to set its position and its other properties.

Editions

sgcMaps is built from the same source for every edition. The Registered and Trial editions include everything; the Trial is for evaluation. The free Community edition leaves out the offline pack: the tile downloader, the MBTiles and PMTiles tile sources, the offline geocoder and the offline router. Everything else is in every edition: the VCL and FireMonkey maps, the tile providers, the HTTP, WMS, WMTS and vector tile sources, all the overlays, the file formats (GeoJSON, GPX, KML, shapefile and WKT), the geometry functions and the online services.

> **Note.** The QuickStart demos use no part of the offline pack, so they compile and run with every edition. The offline demos are listed in chapter 7.

Where the demos are

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

Folder	Project	What it shows
01.HelloMap	`HelloMap.dpr`	The smallest map: centre, zoom and two markers set in the Object Inspector
02.MapMarkers	`MapMarkers.dpr`	Fifteen markers and labels added in code, colours by size, the view fitted to the markers
03.MapShapes	`MapShapes.dpr`	A circle, a rectangle, two polylines (one geodesic) and a polygon
04.MapEvents	`MapEvents.dpr`	Click to add a marker, drag markers, and a log of the map events
05.MapStoreApp	`MapStoreApp.dpr`	A small store locator: list, search, delivery area, dark theme, online layer and image export

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 `Common` folder holds `uQSWorldMap.pas`, the helper that draws the continents, which every demo project lists. The other folders in `Demos\Maps` are larger demos, listed in chapter 7.

Running your first demo

1. In Delphi, choose **File** > **Open Project** and open `Demos\Maps\00.QuickStart\01.HelloMap\HelloMap.dpr`.
2. Press **F9**. The window opens with a map of Europe and two markers.
3. Back in the IDE, open the form, select `Map` and look at `CenterLatitude`, `CenterLongitude`, `Zoom` and `Markers` in the Object Inspector. Those properties are the whole demo.

> **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

This chapter explains the ideas you need before the demos: how a position and a view are described, where the base map comes from, the collections that hold your overlays, the events, the look and the output. Each section names the properties involved, so you can find them in the Object Inspector.

Coordinates, centre and zoom

Every position in sgcMaps is a pair of `Double` values in degrees: the **latitude**, positive to the north of the equator and negative to the south, and the **longitude**, positive to the east of Greenwich and negative to the west. Madrid is latitude 40.4168, longitude −3.7038. Items use the names `Latitude` and `Longitude`, or `CenterLat` and `CenterLon` for a circle, and methods take them as `aLat, aLon`, always in that order.

The view of the map is described by three published properties:

Property	Meaning	Default
`CenterLatitude`, `CenterLongitude`	The point in the middle of the map.	Madrid
`Zoom`	The zoom level. 0 shows the whole world, and every further level doubles the scale: about 3 for a continent, 13 for a city, 18 for a street. `Options.MinZoom` and `Options.MaxZoom` limit the range, 0 to 19 by default.	13

A new map therefore opens on central Madrid at city level. Set the three properties in the Object Inspector for a fixed start view, or move the view from code:

- `SetCenter(aLat, aLon)` centres the map on a point and keeps the zoom.
- `ZoomIn` and `ZoomOut` change the zoom by one level, like the zoom buttons.
- `FitToMarkers(aPaddingPx)` chooses the centre and the zoom that show every marker, with a margin in pixels.
- `FitBounds(aNorthLat, aWestLon, aSouthLat, aEastLon)` shows an area, and `FitToPolyline` shows a line.

The map works in Web Mercator, the projection of web tile servers. You never convert anything yourself: you give positions in degrees and the map places them.

The base map comes from layers

The `Layers` property is a collection of `TsgcMapLayer` items (unit `sgcMaps_Classes`). Each layer says where its tiles come from:

- **`Provider`** names one of the built-in tile servers, such as `mtpOpenStreetMap`, `mtpOpenTopoMap`, `mtpCartoLight`, `mtpEsriWorldImagery` or `mtpMapTilerStreets`. Some providers need a key in `ApiKey`.
- **`TileSourceType`** selects the kind of source when you do not use a provider: `mtsFolder` (a folder of tile images, the default), `mtsMBTiles`, `mtsHTTP` (a server with `URLTemplate`), `mtsPMTiles`, the vector types and `mtsWMS` or `mtsWMTS`.
- **`TileSourcePath`** is the folder or file of a local source, and `URLTemplate` the address of a server.

- `Opacity`, `Visible`, `MinZoom`, `MaxZoom`, `IsBaseLayer`, `CacheFolder` and `Attribution` complete the picture.

This method adds an OpenStreetMap layer from code. `sgcMapApplyProvider` (unit `sgcMaps_Providers`) copies the settings of the provider into the layer, such as its maximum zoom and its attribution:

```
procedure TfrmMain.AddOnlineLayer;
var
  lLayer: TsgcMapLayer;
begin
  lLayer := Map.Layers.Add;
  lLayer.Name := 'OpenStreetMap';
  lLayer.Provider := mtpOpenStreetMap;
  sgcMapApplyProvider(lLayer);
  Map.Options.ShowAttribution := True;
end;
```

And this one adds an offline layer from an MBTiles file next to the executable (Delphi XE7 and later, not in the Community edition):

```
procedure TfrmMain.AddOfflineLayer;
var
  lLayer: TsgcMapLayer;
begin
  lLayer := Map.Layers.Add;
  lLayer.Name := 'Offline';
  lLayer.TileSourceType := mtsMBTiles;
  lLayer.TileSourcePath := ExtractFilePath(ParamStr(0)) + 'europe.mbtiles';
end;
```

Public tile servers have usage policies. OpenStreetMap, for example, asks for a visible credit, which is why the map shows `Options.AttributionText` ("OpenStreetMap contributors" by default) while `Options.ShowAttribution` is `True`, the default. It also allows interactive viewing only: for a production application use a commercial provider, your own tile server or offline tiles.

About the world outline in the demos

Without a layer, the map paints `Options.BackgroundColor` and the overlays, nothing else. That would make the QuickStart demos depend on an internet connection or on a large data file, so they use a small helper instead: `Common\uQSWorldMap.pas`, listed in every demo project as `uQSWorldMap in '..\Common\uQSWorldMap.pas'`.

- `AddWorldOutline(Map)` adds a few dozen coarse polygons, the continents and the main islands, with inland seas as holes, filled with a sand colour. Every polygon has `Tag = CS_QS_WORLD_TAG` (−1) so the demos can find them again.
- `ColorWorldOutline(Map, aLand, aCoast)` recolours them, which demo 05 uses for its dark theme.
- `CS_QS_SEA_COLOR` is the light blue every demo form uses as `Options.BackgroundColor` (the value `15259050` in the DFM).

> **Note.** The outline is demo scaffolding, not a feature of sgcMaps. It is made of ordinary `TsgcMapPolygon` items, the same ones you meet in demo 03, and it is deliberately rough: it looks right at zoom 3 to 5 and crude when you zoom in further. A real application uses a tile layer instead: a provider such as OpenStreetMap, an offline MBTiles or PMTiles file, or vector tiles, as shown by the other demos listed in chapter 7. Demo 05 shows the switch from the outline to an OpenStreetMap layer.

Overlays: the item collections

Everything you draw on the map is an item of a published collection of the map control:

Collection	Item class	Add in code	Described by
`Markers`	`TsgcMapMarker`	`AddMarker(aLat, aLon, aPopup, aTooltip)`	A position
`Labels`	`TsgcMapLabel`	`AddLabel(aLat, aLon, aText)`	A position and a text
`Circles`	`TsgcMapCircle`	`AddCircle(aLat, aLon, aRadiusMeters)`	A centre and a radius in metres
`Rectangles`	`TsgcMapRectangle`	`AddRectangle(aNorthLat, aWestLon, aSouthLat, aEastLon)`	Four edges
`Polylines`	`TsgcMapPolyline`	`AddPolyline`, then `AddPoint`	A list of points, open
`Polygons`	`TsgcMapPolygon`	`AddPolygon`, then `AddPoint`	A list of points, closed and filled
`Heatmaps`	`TsgcMapHeatmap`	through the collection	Weighted points shown as a density map

There are two ways to fill them, and you can mix them freely:

- **At design time**, in the collection editor of the Object Inspector. This suits items described by a few numbers: markers, labels, circles and rectangles. Demos 01 and 03 do this.
- **In code**, with the `Add` methods. This suits data that comes from a database or a file, and polylines and polygons, whose point lists are clearer in code than in the form file. Demos 02 to 05 do this.

Items are ordinary collection items: read them with `Map.Markers[i]` and `Map.Markers.Count`, change their properties and the map repaints, hide them with `Visible := False`, and remove them with the collection methods or, for markers, with `ClearMarkers`.

Markers

A marker is a pin on a position. Its main properties are:

- `Latitude`, `Longitude`: where it stands.
- `PopupText`: the text of the popup that opens when the user clicks the marker. `ShowPopup(aMarker)` opens it from code and `HidePopup` closes it.
- `TooltipText`: the text shown while the mouse is over the marker.
- `Color`: the colour of the pin. `Visible`: hidden markers are neither painted nor clickable.
- `Draggable`: the user can move the marker with the mouse.

- `Tag` (an `Integer`) and `Data` (a `string`): free for you, typically the index or the key of the record behind the marker.
- `IconBitmap` with `AnchorX` and `AnchorY`: your own image instead of the pin.

The `MarkerOptions` property holds the settings shared by all markers: `DefaultWidth`, `DefaultHeight`, `DefaultColor`, the font, and `Clustering`, which groups nearby markers into one numbered circle when the user zooms out.

> **Two details that surprise people.** `AddMarker` is a procedure, it does not return the new marker. The marker it adds is the last item of the collection, `Map.Markers[Map.Markers.Count - 1]`, which is how every demo reaches it. And `AddMarker` copies `MarkerOptions.DefaultColor` into the `Color` of the new marker, while a marker added in the collection editor starts red, the default of `Color`.

Once the markers are in the collection, a loop over it is all you need to change them, for example to colour every visible marker green:

```
procedure TfrmMain.ReadMarkers;
var
  i: Integer;
begin
  for i := 0 to Map.Markers.Count - 1 do
    if Map.Markers[i].Visible then
      Map.Markers[i].Color := clGreen;
end;
```

Shapes and labels

The four shape collections share the same ideas: a fill colour and an opacity from 0 (transparent) to 1 (solid), a stroke colour and width, `Visible`, and `Editable`, which lets the user drag the vertices with the mouse and raises `OnShapeEdited`.

- **Circle:** `CenterLat`, `CenterLon` and `RadiusMeters`. The radius is a real distance, so the circle grows and shrinks with the zoom.
- **Rectangle:** `NorthLatitude`, `WestLongitude`, `SouthLatitude`, `EastLongitude`.
- **Polyline:** `Color`, `Width`, `Dashed`, and `Geodesic`, which draws the shortest path over the globe (a great circle) instead of a straight line on the screen.
- **Polygon:** `FillColor`, `FillOpacity`, `StrokeColor`, `StrokeWidth`, and `Holes` for lakes or courtyards cut out of the area.

The `Add` methods return the new item, so you can set its properties at once. This method adds a 50 km circle around Madrid, zooms to the Iberian Peninsula and switches marker clustering on:

```
procedure TfrmMain.AddCircleAndFit;
var
  lCircle: TsgcMapCircle;
begin
  lCircle := Map.AddCircle(40.4168, -3.7038, 50000);
  lCircle.FillOpacity := 0.2;
  Map.FitBounds(44.0, -10.0, 36.0, 4.0);
  Map.MarkerOptions.Clustering.Enabled := True;
end;
```

A **label** is a text on the map with `Text`, `FontSize`, `FontColor`, and a box behind it (`ShowBackground`, `BackgroundColor`). The top left corner of the label sits on its coordinate, so a label added at the same position as a marker appears to the right of and below the tip of the pin.

Events

The map raises ordinary Delphi events. Double click an event in the Object Inspector and Delphi writes the handler with the right parameters:

Event	Raised when	Parameters
`OnMapClick`	The user clicks the map outside the markers (this also closes an open popup)	`aLat, aLon: Double`, the clicked position
`OnMarkerClick`	The user clicks a marker	`aMarker: TsgcMapMarker`
`OnMarkerDragStart`, `OnMarkerDrag`, `OnMarkerDragEnd`	A draggable marker is moved	`aMarker: TsgcMapMarker`, already at its new position
`OnZoomChanged`	The zoom level changes, by the user or by code	none, read `Zoom`
`OnShapeEdited`	The user changes an editable shape	`aItem: TCollectionItem`, the shape
`OnClusterClick`	The user clicks a cluster of markers	`aCount`, `aLatitude`, `aLongitude` and `var aHandled`
`OnTileError`	A tile cannot be loaded	the layer, the tile and the error text

The marker events use the type `TsgcMapMarkerClickEvent` and the click event `TsgcMapClickEvent`; `TsgcMapMarker` is declared in `sgcMaps_Classes`, so a form with marker events lists that unit next to `sgcMaps_VCL`. Handlers are ordinary methods: you can call them from your own code, which demo 04 uses to add its first markers.

Look and behaviour

- **`Theme`** paints the controls drawn over the map, the zoom buttons, the scale bar, the popups and the layer switcher, in the sgcUI light, dark, custom or high contrast style. It does not change the tiles, the sea or your items.
- **`Options.BackgroundColor`** is the colour painted where there is no tile, which with no layer is the whole map.
- `Options.ShowZoomControls`, `ShowScaleBar`, `ShowAttribution` and `ShowLayerSwitcher` show or hide the controls over the map.
- `Options.DragEnabled` and `ZoomEnabled` switch dragging and zooming off for a fixed map. `Options.RotationEnabled` and `Bearing` rotate it.
- `Options.TileSize` is the size at which tiles are drawn, 256 pixels by default; raise it on high DPI screens, together with `Retina` on the layers.

Images, printing and threads

The map can render itself off screen at any size: `SaveToBMP(aFileName, aWidth, aHeight)` in every Delphi version, and `SaveToPNG` from Delphi 2009. `Print` prints it, `ExportToPDF` writes it to a PDF file or stream, and `PrintSettings` holds the page settings.

The map is a VCL control, so create it and change it in the main thread, as any other control. The online services have `Async` methods that run the request in a thread and raise the result event in the main thread, so their handlers can update the map directly. The QuickStart demos need none of this: everything happens in event handlers of the form.

Summary: who does what

You want to	Use
Choose the start view	`CenterLatitude`, `CenterLongitude`, `Zoom`
Move the view from code	`SetCenter`, `ZoomIn`, `ZoomOut`, `FitToMarkers`, `FitBounds`
Show a base map	`Layers`: a `TsgcMapLayer` with a `Provider` or a `TileSourceType`
Put pins on the map	`Markers`, `AddMarker`, `MarkerOptions`
Draw areas and routes	`Circles`, `Rectangles`, `Polylines`, `Polygons` and their `Add` methods
Write on the map	`Labels`, `AddLabel`
React to the user	`OnMapClick`, `OnMarkerClick`, the drag events, `OnZoomChanged`
Change the look	`Theme`, `Options.BackgroundColor`, the `Options.Show*` properties
Save or print	`SaveToBMP`, `SaveToPNG`, `Print`, `ExportToPDF`

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.HelloMap	`HelloMap.dpr`	Centre and zoom, markers set at design time, the built-in controls
02.MapMarkers	`MapMarkers.dpr`	Markers and labels added in code, marker colours, fitting the view to the data
03.MapShapes	`MapShapes.dpr`	Circles and rectangles at design time, polylines, a geodesic line and a polygon in code
04.MapEvents	`MapEvents.dpr`	Map and marker events, draggable markers, popups from code
05.MapStoreApp	`MapStoreApp.dpr`	Data, a list in sync with the markers, search, a dark theme, a real tile layer, image export

> **Note.** The demos need sgcMaps installed, so the map appears on the SGC Maps palette page and the `sgcUI\Source` folder is on the library path. They use only features that are part of every edition and compile from Delphi 7 on. Every project also lists `..\Common\uQSWorldMap.pas`, the helper that draws the continents (see About the world outline in the demos); keep the `Common` folder next to the demo folders.

4.1 Demo 01: Hello Map

Folder `01.HelloMap`, project `HelloMap.dpr`.

What you will build

The smallest sgcMaps application: a map centred on Europe at continent level, with two markers on Madrid and Paris. The user can drag and zoom the map, hover a marker to see its name and click it to open a popup.

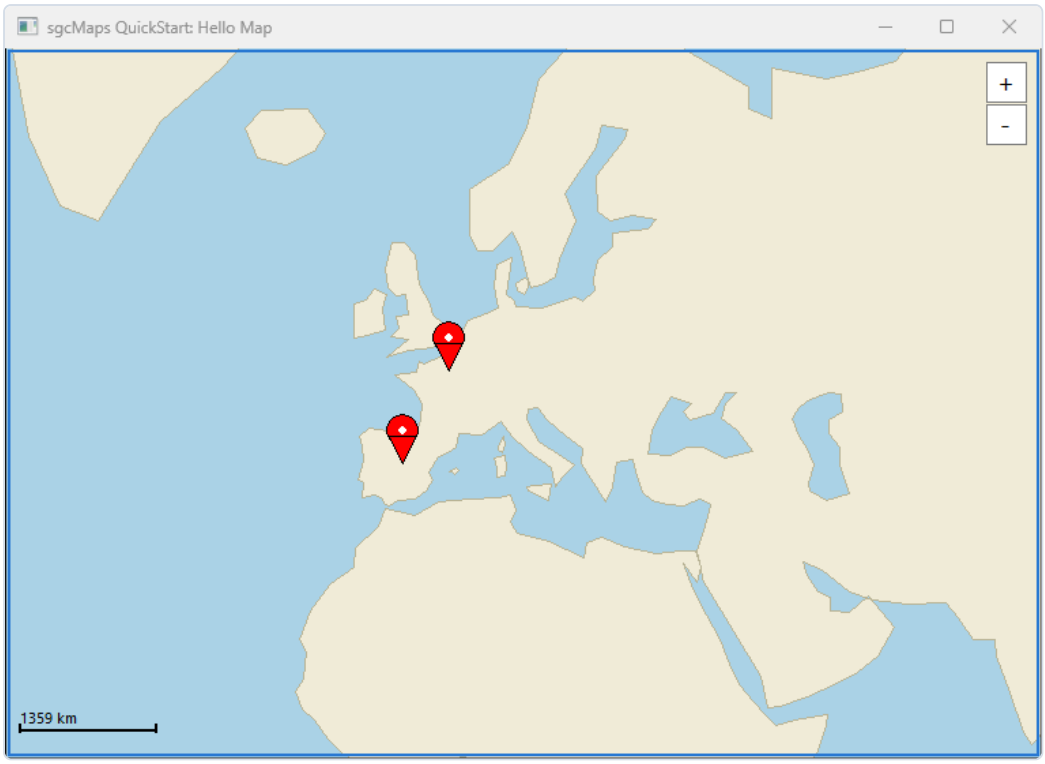


Figure 4.1. Demo 01. The zoom buttons at the top right and the scale bar at the bottom left are part of the control.

Components on the form

Component	Class	Properties you set
`Map`	`TsgcVCLMap`	`Align = alClient`, `CenterLatitude = 46`, `CenterLongitude = 12`, `Zoom = 3`, `Options.ShowAttribution = False`, `Options.BackgroundColor = 15259050` (light sea blue), `Markers` with two items.

Each marker item has a `Latitude`, a `Longitude`, a `PopupText` shown on click and a `TooltipText` shown on hover. The DFM is the one shown in chapter 1. `ShowAttribution` is off because the demo shows no data of a tile provider, so there is nobody to credit.

The code

The `uses` clause of the interface names only the map unit, and the implementation adds the helper unit:

```
uses
  Classes, Controls, Forms,
  // sgc
  sgcMaps_VCL;
```

The whole implementation section is the helper unit and one event handler:

```
implementation

{$R *.dfm}

uses
  uQSWorldMap;

procedure TfrmHelloMap.FormCreate(Sender: TObject);
begin
  { Adds the continents as polygons, see uQSWorldMap.pas. }
  AddWorldOutline(Map);
end;
```

How it works

1. The form loads from its DFM. The map reads its centre, its zoom and the two markers.
2. `FormCreate` calls `AddWorldOutline`, which adds the land polygons to the `Polygons` collection of the map.
3. The map paints the background colour, then the polygons, then the markers, then its own controls: the zoom buttons and the scale bar.
4. Mouse dragging, the wheel and the buttons change the centre and the zoom; the map repaints itself. A click on a marker opens its popup, a click elsewhere closes it.

Try this

- Set `Zoom` to 5 and the centre to the coordinates of your own city. The outline gets rough at that zoom, which is a good reminder that real applications use a tile layer.
- Set `Options.ShowZoomControls` or `Options.ShowScaleBar` to `False`.
- Open the `Markers` collection editor and change the `Color` of one marker.

4.2 Demo 02: Map Markers

Folder `02.MapMarkers`, project `MapMarkers.dpr`.

What you will build

Fifteen European capitals, added in code with a popup that shows the population and a tooltip with the name. Cities with three million inhabitants or more get a red pin, the others a blue one, and every city has a small name label. At the end the map chooses the centre and the zoom that show all fifteen markers.

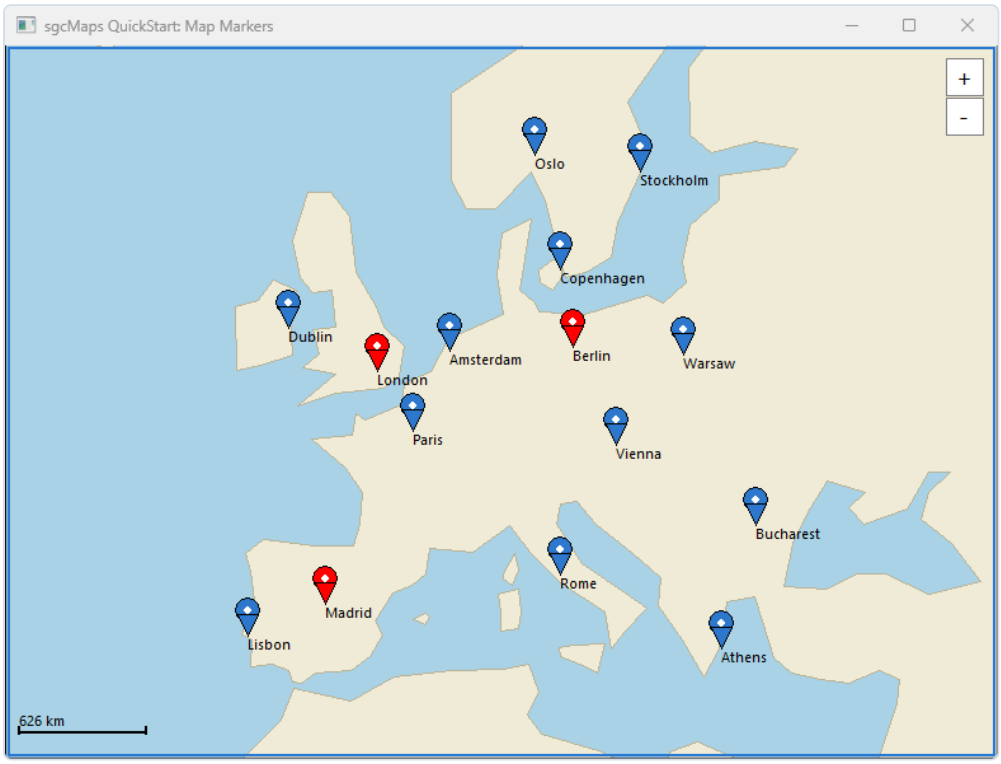


Figure 4.2. Demo 02. London, Berlin and Madrid have red pins; the view was chosen by `FitToMarkers`.

Components on the form

Component	Class	Properties you set
`Map`	`TsgcVCLMap`	`Align = alClient`, `Options.ShowAttribution = False`, `Options.BackgroundColor = 15259050`, `MarkerOptions.DefaultWidth = 20`, `MarkerOptions.DefaultHeight = 30`, `MarkerOptions.DefaultColor = 13662256` (blue).

The code

One private method adds a city: the marker, its colour and its label.

```
procedure TfrmMapMarkers.AddCity(const aName: string;
  aLat, aLon, aMillions: Double);
var
  lMarker: TsgcMapMarker;
  lLabel: TsgcMapLabel;
begin
  { The popup opens when the user clicks the pin, the tooltip shows while
    the mouse is over it. }
  Map.AddMarker(aLat, aLon,
    aName + ': ' + FormatFloat('0.0', aMillions) + ' million inhabitants',
```

```
    aName);

  { The marker just added is the last item of the collection. Big cities
    get a red pin, the others keep MarkerOptions.DefaultColor. }
  lMarker := Map.Markers[Map.Markers.Count - 1];
  if aMillions >= 3 then
    lMarker.Color := SGC_MAP_COLOR_RED;

  { A small label with the name, without the white box behind it. }
  lLabel := Map.AddLabel(aLat, aLon, aName);
  lLabel.FontSize := 8;
  lLabel.ShowBackground := False;
end;
```

`FormCreate` draws the land, adds the cities and fits the view:

```
procedure TfrmMapMarkers.FormCreate(Sender: TObject);
begin
  AddWorldOutline(Map);

  { In a real application this list comes from a database or a file. }
  AddCity('Lisbon', 38.7223, -9.1393, 0.5);
  AddCity('Madrid', 40.4168, -3.7038, 3.3);
  AddCity('Paris', 48.8566, 2.3522, 2.1);
  AddCity('London', 51.5072, -0.1276, 8.9);
  // ...
  AddCity('Bucharest', 44.4268, 26.1025, 1.8);

  { Centre and zoom so the fifteen markers are visible, 40 pixels from the
    edges. }
  Map.FitToMarkers(40);
end;
```

The implementation section uses `sgcMaps_Classes`, where `TsgcMapMarker`, `TsgcMapLabel` and the colour constant `SGC_MAP_COLOR_RED` are declared.

How it works

1. `AddMarker` adds a `TsgcMapMarker` to `Markers` with the position, the popup and the tooltip, and gives it the colour of `MarkerOptions.DefaultColor`.
2. Because `AddMarker` returns nothing, the code takes the new marker from the end of the collection and turns it red for the big cities.
3. `AddLabel` does return its item, a `TsgcMapLabel`, so the code sets the font size and removes the white box. The top left corner of the label sits on the coordinate, which puts the name next to the tip of the pin.
4. `FitToMarkers(40)` computes the area that holds every marker and sets the centre and the zoom so that it fits, with 40 pixels of margin. The centre and the zoom stored in the DFM are replaced.

Try this

- Set `Map.MarkerOptions.Clustering.Enabled := True` before `FitToMarkers`, run and zoom out: nearby markers merge into numbered circles.
- Set `ShowBackground` of the labels to `True` and give them a `BackgroundColor`.
- Add a button that calls `Map.ClearMarkers`. The labels stay, because they are a separate collection.

4.3 Demo 03: Map Shapes

Folder `03.MapShapes`, project `MapShapes.dpr`.

What you will build

The four kinds of shapes on one map: a green circle of 300 km around Paris and an orange rectangle over the Alps, both set at design time; a blue rail route from Lisbon to Amsterdam and a dashed geodesic flight from Dublin to Athens, two polylines added in code; and a semi transparent purple polygon over Denmark and southern Sweden. A label names each shape.

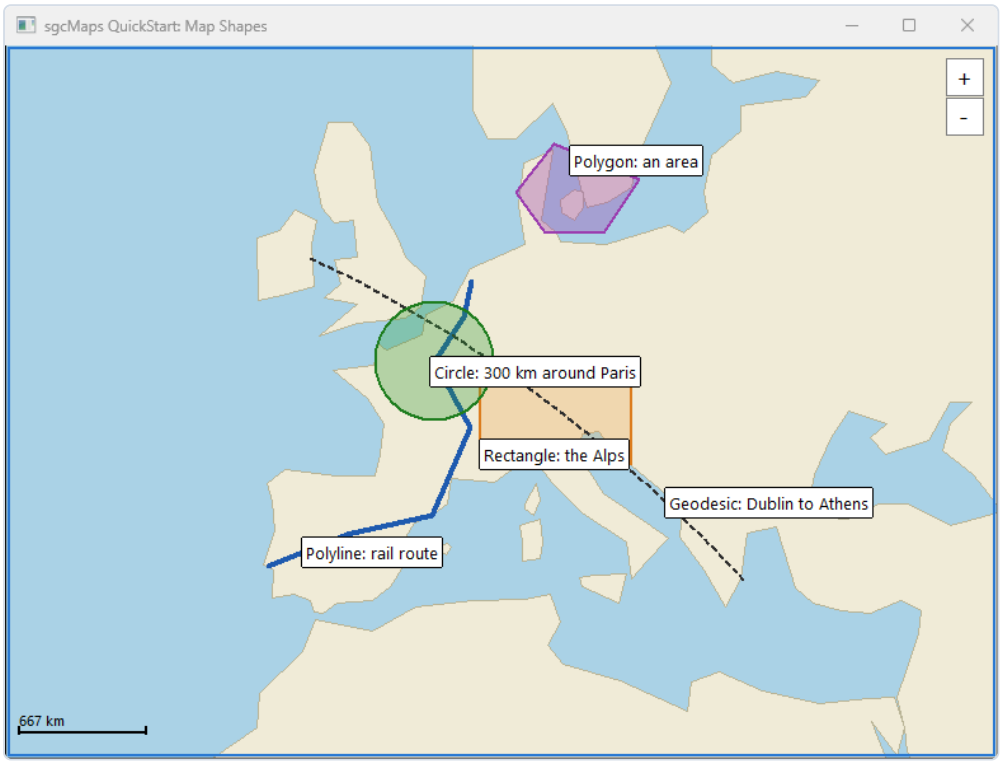


Figure 4.3. Demo 03. The labels keep their default white box, and each one hangs from its coordinate by its top left corner.

Components on the form

Component	Class	Properties you set
`Map`	`TsgcVCLMap`	`Align = alClient`, `CenterLatitude = 47`, `CenterLongitude = 7`, `Zoom = 4`, `Options.ShowAttribution = False`, `Options.BackgroundColor = 15259050`, one item in `Circles` and one in `Rectangles`.

The circle and the rectangle are described by a few numbers, so they are set in the collection editors:

```
    Circles = <
      item
        CenterLat = 48.856600000000000000
        CenterLon = 2.352200000000000000
        RadiusMeters = 300000.000000000000000000
        FillColor = 3186736
        StrokeColor = 2129952
        FillOpacity = 0.300000000000000000
      end>
    Rectangles = <
      item
```

```
        NorthLatitude = 48.000000000000000000
        WestLongitude = 5.500000000000000000
        SouthLatitude = 44.000000000000000000
        EastLongitude = 16.000000000000000000
        FillColor = 4235504
        StrokeColor = 2130144
        FillOpacity = 0.250000000000000000
      end>
```

The code

The polylines and the polygon are lists of points, which read better in code than in the form file:

```
procedure TfrmMapShapes.FormCreate(Sender: TObject);
var
  lLine: TsgcMapPolyline;
  lArea: TsgcMapPolygon;
begin
  AddWorldOutline(Map);

  { A rail route: an open line through seven cities. }
  lLine := Map.AddPolyline;
  lLine.Color := $00B05A1E;
  lLine.Width := 4;
  lLine.AddPoint(38.7223, -9.1393); // Lisbon
  lLine.AddPoint(40.4168, -3.7038); // Madrid
  // ...
  lLine.AddPoint(52.3676, 4.9041); // Amsterdam

  { A flight: only two points, but Geodesic draws the great circle between
    them, and Dashed draws it as a dashed line. }
  lLine := Map.AddPolyline;
  lLine.Color := $00303030;
  lLine.Width := 2;
  lLine.Dashed := True;
  lLine.Geodesic := True;
  lLine.AddPoint(53.3498, -6.2603); // Dublin
  lLine.AddPoint(37.9838, 23.7275); // Athens

  { An area: a closed polygon, the last point joins the first one. }
  lArea := Map.AddPolygon;
  lArea.FillColor := $00B0409C;
  lArea.FillOpacity := 0.35;
  lArea.StrokeColor := $00B0409C;
  lArea.StrokeWidth := 2;
  lArea.AddPoint(56.0, 8.0);
  lArea.AddPoint(57.8, 10.6);
  lArea.AddPoint(56.5, 16.5);
  lArea.AddPoint(54.4, 14.0);
  lArea.AddPoint(54.4, 10.0);

  { Labels name the shapes. }
  Map.AddLabel(48.8566, 2.3522, 'Circle: 300 km around Paris');
  Map.AddLabel(45.0, 5.8, 'Rectangle: the Alps');
  Map.AddLabel(40.0, -6.5, 'Polyline: rail route');
  Map.AddLabel(42.6, 18.6, 'Geodesic: Dublin to Athens');
  Map.AddLabel(57.6, 12.0, 'Polygon: an area');
end;
```

Colours in code are Delphi `TColor` values in the `$00BBGGRR` order: `$00B05A1E` is a blue, not an orange.

How it works

1. The form loads the circle and the rectangle from the DFM. The circle radius is in metres, so it stays 300 km wide on the ground at every zoom level.
2. `AddPolyline` returns an empty `TsgcMapPolyline`; each `AddPoint(aLat, aLon)` adds a vertex. A polyline stays open.
3. The second polyline has only two points, but `Geodesic` makes the map draw the shortest path over the globe between them, which on a Mercator map is a curve.
4. `AddPolygon` returns a `TsgcMapPolygon`. It is closed automatically and filled with `FillColor` at `FillOpacity`, so the land shows through.
5. The labels are added last, with their default white box.

Try this

- Set `lArea.Editable := True`, run, and drag the vertices of the polygon. Handle `OnShapeEdited` to be told when the user changes it.
- Call `Map.FitToPolyline(lLine)` after the first polyline to zoom to the rail route.
- Cut a hole in the polygon with its `Holes` property, the way the world outline cuts out the Black Sea and the Caspian Sea.

4.4 Demo 04: Map Events

Folder `04.MapEvents`, project `MapEvents.dpr`.

What you will build

A map that reacts to the user. A click on the map adds a draggable marker at that point. Clicking a marker, dropping a dragged marker and changing the zoom are written to a log on the right, and the status bar shows the zoom level and the number of markers. Three markers are already there when the window opens, and the popup of the third one is open.

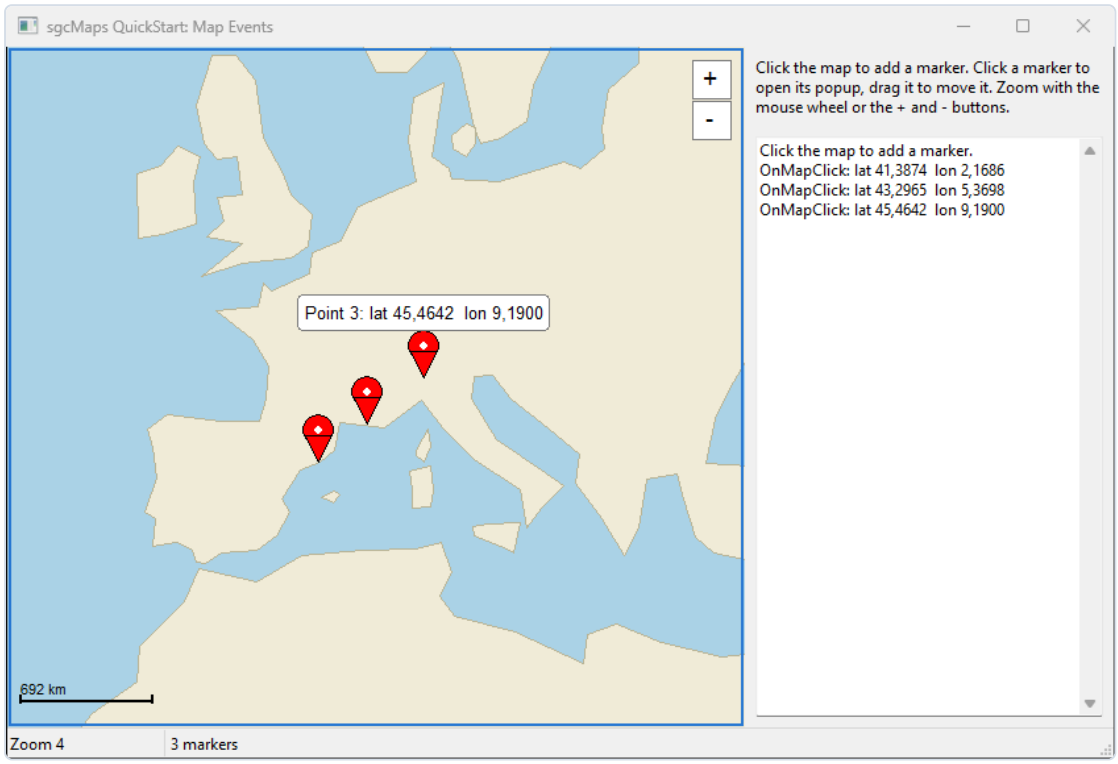


Figure 4.4. Demo 04. The log shows the events raised so far, and the third marker's popup was opened from code.

Components on the form

Component	Class	Properties you set
`Map`	`TsgcVCLMap`	`Align = alClient`, `CenterLatitude = 45`, `CenterLongitude = 6`, `Zoom = 4`, the usual `Options`, and four events: `OnMapClick`, `OnMarkerClick`, `OnMarkerDragEnd`, `OnZoomChanged`.
`pnlRight`	`TPanel`	`Align = alRight`, holds the two controls below.
`lblHelp`	`TLabel`	`Align = alTop`, `WordWrap = True`, a short instruction.
`memLog`	`TMemo`	`Align = alClient`, `ReadOnly = True`, the event log.
`StatusBar`	`TStatusBar`	Two panels: the zoom and the marker count.

The events are connected in the Object Inspector, so the DFM lists them on the map:

```
OnMarkerClick = MapMarkerClick
OnMarkerDragEnd = MapMarkerDragEnd
OnMapClick = MapMapClick
OnZoomChanged = MapZoomChanged
```

The code

The marker events pass a `TsgcMapMarker`, so the interface `uses` clause lists `sgcMaps_Classes` next to `sgcMaps_VCL`:

```
uses
  Classes, Controls, Forms, StdCtrls, ComCtrls, ExtCtrls,
  // sgc
  sgcMaps_Classes, sgcMaps_VCL;

type
  TfrmMapEvents = class(TForm)
    // ...
    procedure FormCreate(Sender: TObject);
    procedure MapMapClick(Sender: TObject; aLat, aLon: Double);
    procedure MapMarkerClick(Sender: TObject; aMarker: TsgcMapMarker);
    procedure MapMarkerDragEnd(Sender: TObject; aMarker: TsgcMapMarker);
    procedure MapZoomChanged(Sender: TObject);
  private
    procedure Log(const aText: string);
    procedure UpdateStatus;
  end;
```

`FormCreate` seeds three markers by calling the click handler, then opens a popup from code:

```
procedure TfrmMapEvents.FormCreate(Sender: TObject);
begin
  AddWorldOutline(Map);
  Log('Click the map to add a marker.');
  { The same handler the map calls when the user clicks on the map. }
  MapMapClick(Map, 41.3874, 2.1686); // Barcelona
  MapMapClick(Map, 43.2965, 5.3698); // Marseille
  MapMapClick(Map, 45.4642, 9.1900); // Milan
  { ShowPopup opens the popup of a marker from code. }
  Map.ShowPopup(Map.Markers[2]);
  UpdateStatus;
end;
```

The four event handlers:

```
procedure TfrmMapEvents.MapMapClick(Sender: TObject; aLat, aLon: Double);
var
  lMarker: TsgcMapMarker;
begin
  Map.AddMarker(aLat, aLon, 'Point ' + IntToStr(Map.Markers.Count + 1) +
    ': ' + FormatLatLon(aLat, aLon), 'Drag me');
  lMarker := Map.Markers[Map.Markers.Count - 1];
  { Draggable markers can be moved with the mouse; OnMarkerDragEnd tells
    where they were dropped. }
  lMarker.Draggable := True;
  Log('OnMapClick: ' + FormatLatLon(aLat, aLon));
  UpdateStatus;
end;

procedure TfrmMapEvents.MapMarkerClick(Sender: TObject; aMarker: TsgcMapMarker);
begin
  { The popup opens by itself; the event is for your own reaction, such as
    showing details of the record behind the marker. }
  Log('OnMarkerClick: marker ' + IntToStr(aMarker.Index + 1) + ' at ' +
    FormatLatLon(aMarker.Latitude, aMarker.Longitude));
```

```
end;

procedure TfrmMapEvents.MapMarkerDragEnd(Sender: TObject;
  aMarker: TsgcMapMarker);
begin
  { Latitude and Longitude already hold the new position. }
  aMarker.PopupText := 'Point ' + IntToStr(aMarker.Index + 1) + ': ' +
    FormatLatLon(aMarker.Latitude, aMarker.Longitude);
  Log('OnMarkerDragEnd: marker ' + IntToStr(aMarker.Index + 1) +
    ' moved to ' + FormatLatLon(aMarker.Latitude, aMarker.Longitude));
end;

procedure TfrmMapEvents.MapZoomChanged(Sender: TObject);
begin
  Log('OnZoomChanged: zoom ' + IntToStr(Map.Zoom));
  UpdateStatus;
end;
```

The helpers are plain Delphi: `FormatLatLon` formats a position with `FormatFloat('0.0000', ...)`, `Log` adds a line to the memo and `UpdateStatus` writes `Map.Zoom` and `Map.Markers.Count` to the status bar.

How it works

1. A click on an empty part of the map raises `OnMapClick` with the latitude and longitude under the mouse. The handler adds a marker there and makes it draggable.
2. A click on a marker raises `OnMarkerClick` instead, and the map opens the popup of the marker by itself. The handler only logs; in a real application it would show the record behind the marker.
3. Dragging a draggable marker raises `OnMarkerDragStart`, `OnMarkerDrag` while it moves, and `OnMarkerDragEnd` when it is dropped. By then `Latitude` and `Longitude` already hold the new position, so the handler rewrites the popup text.
4. Every zoom change, from the wheel, the buttons or code, raises `OnZoomChanged`.
5. `aMarker.Index` is the position of the marker in the `Markers` collection, a property every collection item has.

Try this

- Handle `OnMarkerDrag` and show the live coordinates of the marker in the status bar while it moves.
- Set `Options.DragEnabled` or `Options.ZoomEnabled` to `False` and see which events still arrive.
- In `MapMarkerClick`, set `aMarker.Visible := False`: clicked markers disappear.

4.5 Demo 05: Store Locator

Folder `05.MapStoreApp`, project `MapStoreApp.dpr`.

What you will build

A small but complete application: a store locator for twelve stores in Spain, Portugal and the south west of France. The list on the left and the markers on the map stay in sync both ways, a search box filters the list and the markers together, the selected store shows a 60 km delivery area, a check box switches to a dark theme, another one replaces the world outline with a real OpenStreetMap layer, and a button saves the map as an image.

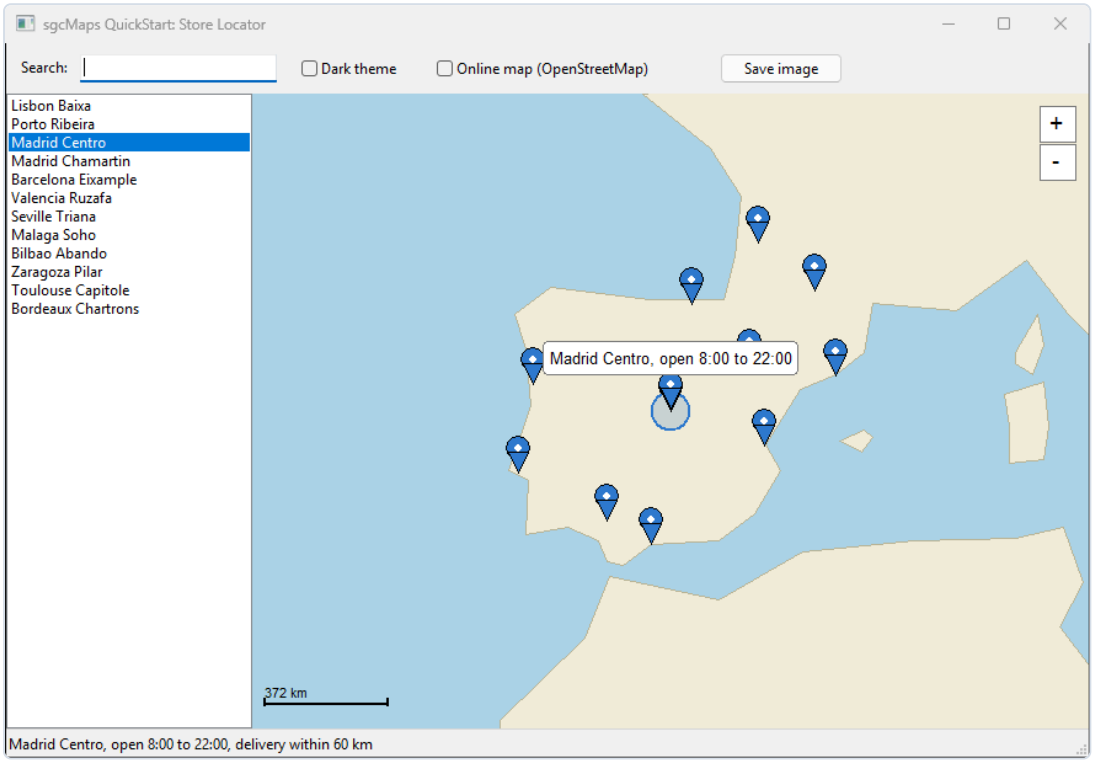


Figure 4.5. Demo 05 at start up: Madrid Centro is selected, its popup is open and its delivery area is shown.

Components on the form

Component	Class	Properties you set
`pnlTop`	`TPanel`	`Align = alTop`, holds `lblSearch`, `edtSearch`, `chkDark`, `chkOnline` and `btnSave`.
`edtSearch`	`TEdit`	`OnChange = edtSearchChange`.
`chkDark`, `chkOnline`	`TCheckBox`	`OnClick`: the dark theme and the online map.
`btnSave`	`TButton`	`OnClick = btnSaveClick`.
`lstStores`	`TListBox`	`Align = alLeft`, `OnClick = lstStoresClick`.
`Map`	`TsgcVCLMap`	`Align = alClient`, `MarkerOptions.DefaultColor = 13662256` (blue), one item in `Circles` (`RadiusMeters = 60000`, `FillOpacity = 0.2`, `Visible = False`), `OnMarkerClick = MapMarkerClick`.
`StatusBar`	`TStatusBar`	`SimplePanel = True`.

The delivery circle is created hidden at design time and moved to the selected store in code, so the application never adds or removes circles.

The code

The stores are a typed constant, a stand in for a database:

```
type
  TStore = record
    Name: string;
    Hours: string;
    Lat: Double;
    Lon: Double;
  end;

const
  { In a real application the stores come from a database. }
  STORES: array[0..11] of TStore = (
    (Name: 'Lisbon Baixa'; Hours: '8:00 to 20:00'; Lat: 38.7110; Lon: -9.1366),
    (Name: 'Porto Ribeira'; Hours: '9:00 to 21:00'; Lat: 41.1405; Lon: -8.6131),
    // ...
    (Name: 'Bordeaux Chartrons'; Hours: '9:00 to 19:00'; Lat: 44.8550; Lon: -0.5700));
```

`FormCreate` adds one marker per store, in the same order as the table, and remembers the index in `Tag`:

```
procedure TfrmMapStoreApp.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  AddWorldOutline(Map);

  { One marker per store, in the same order as the table, so the marker of
    store i is Map.Markers[i]. Tag keeps the index too. }
  for i := Low(STORES) to High(STORES) do
  begin
    Map.AddMarker(STORES[i].Lat, STORES[i].Lon,
      STORES[i].Name + ', open ' + STORES[i].Hours, STORES[i].Name);
    Map.Markers[i].Tag := i;
  end;
  FillList;
  Map.FitToMarkers(40);

  { Start with a store selected, so the delivery area is visible. }
  SelectStore(2);
end;
```

`FillList` applies the search to the list and to the markers at the same time:

```
    for i := Low(STORES) to High(STORES) do
    begin
      lMatch := (lFilter = '') or (Pos(lFilter, LowerCase(STORES[i].Name)) > 0);
      { A hidden marker is neither painted nor clickable. }
      Map.Markers[i].Visible := lMatch;
      if lMatch then
        lstStores.Items.Add(STORES[i].Name);
    end;
```

`SelectStore` is called from the list and from the map, and keeps both in step:

```
procedure TfrmMapStoreApp.SelectStore(aIndex: Integer);
var
  lMarker: TsgcMapMarker;
begin
  lMarker := Map.Markers[aIndex];

  { Move the delivery area, set at design time in Circles[0]. }
  Map.Circles[0].CenterLat := lMarker.Latitude;
  Map.Circles[0].CenterLon := lMarker.Longitude;
  Map.Circles[0].Visible := True;

  { Centre the map on the store and open its popup. }
  Map.SetCenter(lMarker.Latitude, lMarker.Longitude);
  Map.ShowPopup(lMarker);

  { Select the list item that points to the same store. }
  lstStores.ItemIndex := lstStores.Items.IndexOf(STORES[aIndex].Name);
  StatusBar.SimpleText := STORES[aIndex].Name + ', open ' +
    STORES[aIndex].Hours + ', delivery within 60 km';
end;

// ...

procedure TfrmMapStoreApp.MapMarkerClick(Sender: TObject;
  aMarker: TsgcMapMarker);
begin
  SelectStore(aMarker.Tag);
end;
```

The dark theme changes three things, because `Theme` paints only the controls over the map (it comes from the unit `sgcUICore_Theme`):

```
procedure TfrmMapStoreApp.chkDarkClick(Sender: TObject);
begin
  { Theme paints the zoom buttons, the scale bar and the popups. The sea is
    Options.BackgroundColor and the land is the outline polygons, so they
    change colour too. }
  if chkDark.Checked then
  begin
    Map.Theme := utDark;
    Map.Options.BackgroundColor := $00402C1E;
    ColorWorldOutline(Map, $00585048, $00787068);
  end
  else
  begin
    Map.Theme := utLight;
    Map.Options.BackgroundColor := CS_QS_SEA_COLOR;
    ColorWorldOutline(Map, CS_QS_LAND_COLOR, CS_QS_COAST_COLOR);
  end;
end;
```

The online map is what a real application does from the start: a tile layer with a provider. The outline polygons would cover the tiles, so they are hidden by their `Tag`:

```
procedure TfrmMapStoreApp.SetWorldOutlineVisible(aVisible: Boolean);
var
  i: Integer;
begin
  for i := 0 to Map.Polygons.Count - 1 do
```

```
    if Map.Polygons[i].Tag = CS_QS_WORLD_TAG then
      Map.Polygons[i].Visible := aVisible;
end;

procedure TfrmMapStoreApp.chkOnlineClick(Sender: TObject);
var
  lLayer: TsgcMapLayer;
begin
  { A tile layer is the usual base map of a real application. With a
    Provider the layer downloads its tiles over HTTP by itself. The
    outline polygons would cover the tiles, so they are hidden. }
  Map.Layers.Clear;
  if chkOnline.Checked then
  begin
    lLayer := Map.Layers.Add;
    lLayer.Name := 'OpenStreetMap';
    lLayer.Provider := mtpOpenStreetMap;
  end;
  SetWorldOutlineVisible(not chkOnline.Checked);
  { OpenStreetMap requires its credit on the map. }
  Map.Options.ShowAttribution := chkOnline.Checked;
end;
```

And the image export renders the map off screen at a fixed size:

```
procedure TfrmMapStoreApp.btnSaveClick(Sender: TObject);
var
  lFile: string;
begin
  { SaveToBMP renders the map off screen at any size; Delphi 2009 and later
    also have SaveToPNG. }
  lFile := ExtractFilePath(ParamStr(0)) + 'stores.bmp';
  Map.SaveToBMP(lFile, 1200, 800);
  StatusBar.SimpleText := 'Saved ' + lFile;
end;
```

How it works

1. The markers are added in the order of the `STORES` table, so store `i` is always `Map.Markers[i]`, and `Tag` holds the same index. That one rule links the data, the list and the map.
2. A click in the list finds the store by name (`StoreIndex`) and calls `SelectStore`; a click on a marker calls `SelectStore(aMarker.Tag)`. `SelectStore` moves the circle, centres the map, opens the popup and selects the list item, so both paths end in the same state.
3. The search box hides the markers that do not match with `Visible := False` and rebuilds the list. The marker indexes never change, because no marker is deleted.
4. The dark theme recolours the map controls with `Theme`, and the sea and the land with `Options.BackgroundColor` and `ColorWorldOutline`.
5. The online check box adds a `TsgcMapLayer` with `Provider := mtpOpenStreetMap`. The layer downloads the tiles itself; the code only hides the outline and switches the attribution on, as OpenStreetMap requires.
6. `SaveToBMP` paints the same map, layers and items included, into a 1200 by 800 bitmap file next to the executable.

> **The online map needs internet and Delphi 2009 or later.** Tiles from a server are not available in Delphi 7 and 2007, and from Delphi 2009 to XE7 https downloads need the OpenSSL 1.0.2 libraries next to the executable. See The base map in chapter 6.

Try this

- Change the provider to `mtpOpenTopoMap`.
- Add a second layer with another provider and set `Map.Options.ShowLayerSwitcher := True`, so the user can choose the base map.
- On Delphi 2009 or later, save a PNG with `Map.SaveToPNG`, or print the map with `Map.Print`.

5. Main components

The QuickStart demos use one component, the map, but the SGC Maps palette page has more. This chapter introduces the map control, the classes you meet in code, and the non visual components you will want later.

Every class has its own help topic in the Maps section of the sgcUI help, such as `TsgcVCLMap`, with the properties, methods and events.

Reference table

Class	Palette page	What it is for	Key members or notes
`TsgcVCLMap`	SGC Maps	The VCL map	`CenterLatitude`, `CenterLongitude`, `Zoom`, `Layers`, `Markers`, `Options`, `Theme`, `OnMapClick`, `OnMarkerClick`
`TsgcFMXMap`	SGC Maps (FireMonkey)	The FireMonkey map	The same model as the VCL map
`TsgcMapLayer`	none, an item of `Layers`	One source of tiles	`Provider`, `TileSourceType`, `TileSourcePath`, `URLTemplate`, `ApiKey`, `Opacity`
`TsgcMapMarker`	none, an item of `Markers`	A pin	`Latitude`, `Longitude`, `PopupText`, `TooltipText`, `Color`, `Draggable`, `Tag`
`TsgcMapLabel`	none, an item of `Labels`	A text	`Text`, `FontSize`, `FontColor`, `ShowBackground`
`TsgcMapCircle`, `TsgcMapRectangle`	none, items of `Circles` and `Rectangles`	Simple areas	`RadiusMeters`, the four edges, `FillColor`, `FillOpacity`
`TsgcMapPolyline`, `TsgcMapPolygon`	none, items of `Polylines` and `Polygons`	Routes and areas	`AddPoint`, `Geodesic`, `Dashed`, `Holes`, `Editable`
`TsgcMapGeocoder`, `TsgcMapPlaces`	SGC Maps	Addresses and places, online	Delphi XE8 and later
`TsgcMapDirections`, `TsgcMapRouteCalculator`	SGC Maps	Routes	Delphi XE8 (directions) and XE2 (route calculator) and later
`TsgcMapElevation`, `TsgcMapTimeZone`	SGC Maps	Elevation and time zone of a point	Delphi XE8 and later
`TsgcMapOfflineGeocoder`, `TsgcMapGeoIndexBuilder`, `TsgcMapOfflineRouter`	SGC Maps	Search and routing with no network	Registered and Trial editions
`TsgcMapLocation`	SGC Maps	The position of the device	Delphi XE6 and later

The map control

TsgcVCLMap

The control you drop first, in the unit `sgcMaps_VCL`. It holds the view, the layers and every overlay collection, and draws them all.

- **View:** `CenterLatitude`, `CenterLongitude`, `Zoom`; `SetCenter`, `ZoomIn`, `ZoomOut`, `FitToMarkers`, `FitBounds`, `FitToPolyline`.
- **Content:** `Layers`, `Markers`, `Labels`, `Circles`, `Rectangles`, `Polylines`, `Polygons`, `Heatmaps`, and `MarkerOptions` with the clustering settings.
- **Look:** `Theme`, `Options.BackgroundColor`, `Options.ShowZoomControls`, `ShowScaleBar`, `ShowAttribution`, `ShowLayerSwitcher`, `TileSize`.
- **Behaviour:** `Options.DragEnabled`, `ZoomEnabled`, `RotationEnabled`, `Bearing`, `MinZoom`, `MaxZoom`.
- **Events:** `OnMapClick`, `OnMarkerClick`, `OnMarkerDragStart`, `OnMarkerDrag`, `OnMarkerDragEnd`, `OnZoomChanged`, `OnShapeEdited`, `OnClusterClick`, `OnLayerChanged`, `OnBearingChanged`, `OnTileError`.
- **Output:** `ShowPopup`, `HidePopup`, `SaveToBMP`, `SaveToPNG`, `Print`, `ExportToPDF`, `PrintSettings`.

TsgcFMXMap

The FireMonkey map, in the unit `sgcMaps_FMX`, from Delphi XE6 on, for Windows and, from Delphi 10.4, for macOS, iOS and Android. It shares the item collections, the layers and the tile sources of the framework neutral core with the VCL map, so the ideas of this guide carry over. Chapter 7 lists its demos.

Layers and items

The layers and the items are declared in `sgcMaps_Classes`. Add that unit to the `uses` clause as soon as your code declares a variable of one of these classes or handles a marker event, as demos 02 to 05 do.

- **`TsgcMapLayer`** chooses a provider or a tile source, with its own cache folder, opacity and zoom range. Several layers can be stacked, for example a base map and a semi transparent overlay from a WMS server.
- **`TsgcMapMarker`** is the item you will use most: a position, a popup, a tooltip, a colour or an image, and a `Tag` or `Data` that links it to your record.
- **The shapes** share fill, stroke, opacity, `Visible` and `Editable`. Use circles for distances, rectangles for bounding boxes, polylines for routes and tracks, and polygons for areas.
- **`TsgcMapLabel`** writes text on the map, anchored by its top left corner.

The core also reads and writes the usual geographic file formats (GeoJSON, GPX, KML, shapefiles and WKT) into these collections; see the File formats topic in the help.

The online services

Five components ask online services for data and return the result in an event or as the result of a call: `TsgcMapGeocoder` (address to coordinates and back), `TsgcMapPlaces` (search and suggestions while typing), `TsgcMapDirections` (a route between points), `TsgcMapElevation` and `TsgcMapTimeZone`. They need Delphi XE8 or later. Each one offers a choice of providers, some of which need an API key, and `Async` methods that keep the user interface responsive. `TsgcMapRouteCalculator`, from Delphi XE2, calculates routes as well.

The results are positions and lists of points, so they go straight into the map: a marker for a found address, a polyline for a route.

Offline data and location

For applications that must work with no network, the offline pack (Registered and Trial editions) adds MBTiles and PMTiles tile files, a tile downloader to prepare them from a server whose terms allow it, `TsgcMapOfflineGeocoder` with `TsgcMapGeoIndexBuilder` to search places in a local database, and `TsgcMapOfflineRouter` to calculate routes from a local graph. The tools in `Tools\Maps` build those data files.

`TsgcMapLocation`, from Delphi XE6, reports the position of the device, which is mostly useful with the FireMonkey map on phones and tablets.

6. Common problems

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

The base map

The map is a flat colour

What you see. Your markers and shapes are there, but behind them there is no land, no sea and no roads, only a plain colour.

Why. sgcMaps has no built-in world data. With an empty `Layers` collection, which is how a new map starts, the map paints `Options.BackgroundColor` and your items only. The QuickStart demos hide this with the world outline helper.

Fix. Add a `TsgcMapLayer` to `Layers`: a `Provider` such as `mtpOpenStreetMap`, or a `TileSourceType` with a `TileSourcePath` for offline files, as shown in The base map comes from layers. Do not copy `uQSWorldMap.pas` into a real application as a base map: it is a rough sketch meant for the demos.

The layer shows no tiles

What you see. A layer is set, but the map stays empty or shows the text "Tile source not available" with a reason.

Why. Several things can stop the tiles:

- A layer with no `Provider` keeps `TileSourceType = mtsFolder`, the default, and a `TileSourcePath` that does not exist gives "Tile source directory not found".
- Tiles from a server need Delphi 2009 or later, and MBTiles files Delphi XE7 or later. On older versions the layer reports that the tile source type is not available in this compiler.
- From Delphi 2009 to XE7 the tiles are downloaded with Indy, and https needs the OpenSSL 1.0.2 libraries (`libeay32.dll` and `ssleay32.dll`) next to the executable. From Delphi XE8 the map uses `THTTPClient` and needs no DLL.
- The MapTiler, Thunderforest and Stadia providers need an API key in `ApiKey`.
- No internet connection, or a firewall that blocks the tile server.

Fix. Handle `OnTileError`: it is raised once for each tile that fails, with the layer, the tile and the error text, such as an HTTP status or the missing OpenSSL libraries. A tile the server does not have (HTTP 404) is not an error.

OpenStreetMap tiles in a production application

What you see. Everything works during development, but you are not sure you may ship it.

Why. The public OpenStreetMap tile server allows interactive viewing only: no bulk downloading, no prefetching of areas, and a visible attribution. An application that breaks the policy can be blocked without warning.

Fix. Keep `Options.ShowAttribution` on with OpenStreetMap, as demo 05 does. For a production application use a commercial provider, your own tile server, or offline tiles.

The world outline looks crude

What you see. In the QuickStart demos, the coasts become angular when you zoom in.

Why. The outline is a few dozen coarse polygons, made to run the demos offline. It is meant for zoom 3 to 5.

Fix. Nothing to fix in the demos. In your own application use a tile layer, which brings the detail of every zoom level.

Markers, labels and colours

AddMarker does not return the marker

What you see. `lMarker := Map.AddMarker(...)` does not compile.

Why. `AddMarker` is a procedure. The other `Add` methods (`AddLabel` , `AddCircle` , `AddRectangle` , `AddPolyline` , `AddPolygon`) do return their item.

Fix. Call `AddMarker` , then take the new marker from the end of the collection: `lMarker := Map.Markers[Map.Markers.Count - 1];` . Every demo from 02 on does this.

Markers have the wrong colour

What you see. Markers added in the collection editor are red although `MarkerOptions.DefaultColor` is blue, or changing `DefaultColor` does not recolour the markers you already have.

Why. `AddMarker` copies `MarkerOptions.DefaultColor` into the `Color` of each marker it adds. A marker created in the collection editor starts with the default of its `Color` property, which is red.

Fix. Set `Color` on each design time marker, or loop over `Markers` and set `Color` when you want to recolour existing markers, as in Markers.

A label is not where I put it

What you see. A label added at the position of a marker appears below and to the right of it, not centred on it.

Why. The top left corner of a label sits on its coordinate.

Fix. That is usually what you want next to a pin. To place it elsewhere, give the label a slightly different latitude or longitude, as demo 03 does for its shape names.

The dark theme does not darken the map

What you see. With `Theme := utDark` the zoom buttons and the popups turn dark, but the sea, the tiles and your shapes do not.

Why. `Theme` paints only the controls drawn over the map: the zoom buttons, the scale bar, the popups and the layer switcher.

Fix. Change `Options.BackgroundColor` and the colours of your items yourself, as demo 05 does, and for a dark base map choose a dark tile provider such as `mtpCartoDark` or `mtpStadiaAlidadeSmoothDark` .

I need the coordinates under the mouse

What you see. You want to show the position under the mouse while it moves, and find no public method that converts a screen point to a latitude and longitude.

Why. The conversion methods of the control are protected.

Fix. Use the events that already carry a position: `OnMapClick` gives the clicked latitude and longitude, and the drag events give a marker whose `Latitude` and `Longitude` follow the mouse, as in demo 04.

Versions, editions and output

SaveToPNG is not found

What you see. `Map.SaveToPNG` gives "undeclared identifier" on Delphi 7 or 2007.

Why. `SaveToPNG` exists from Delphi 2009 on.

Fix. Use `SaveToBMP`, which works in every version, as demo 05 does.

A component is missing from the palette

What you see. The SGC Maps page shows the map but not the geocoder, the router or the offline components.

Why. Those components depend on your Delphi version: the online services need Delphi XE8, the route calculator XE2, the location component XE6 and the offline geocoder XE7. The offline components are also not part of the Community edition.

Fix. Check the minimum versions in The palette page. The map itself and everything the QuickStart demos use are on the palette in every version and edition.

The map looks small or blurry on a high DPI screen

What you see. On a monitor scaled to 150 % or 200 %, the tiles look small or soft.

Why. The VCL map draws its tiles at `Options.TileSize` pixels, 256 by default, and does not scale them with the DPI of the monitor by itself.

Fix. Set `Options.TileSize` from the pixels per inch of the screen, and set `Retina` on the layers whose provider supports 512 pixel tiles. The High DPI section of the troubleshooting topic in the help shows the code.

The answer of a service never arrives

What you see. In a console application or a service, the result event of an `Async` call is never raised.

Why. The services raise their events in the main thread with `TThread.Queue`, which needs a message loop. A VCL or FireMonkey application has one; a console application does not.

Fix. In a console application call `CheckSynchronize` in a loop until the answer arrives, and keep the component alive until then. A VCL application, like the demos, needs nothing.

7. Where next

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

The other Delphi demos

The other folders in `Demos\Maps` are complete applications. Each one has a topic in the help, in the Demos section of sgcMaps, that explains what it shows and what it needs. They use real tile layers instead of the world outline.

Demo	What it shows
`01.OfflineMap`	A map whose base layer comes from a folder of tiles, an MBTiles file or an online server, with markers, shapes, GeoJSON, GPX and KML import and an event log. Registered and Trial editions.
`02.MapFeatures`	The drawing and interaction features of the VCL map, one tab each: layers and providers, clustering 2000 markers, a heatmap, editable shapes and rotation.
`03.Services`	Every online service on one form, each tab with a provider list and an API key box, with the result drawn on the map. Delphi XE8.
`04.OfflineServices`	Search and routing with no network connection. Delphi XE7, Registered and Trial editions.
`05.StaticRender`	A console application that creates a map with no window, loads the tiles of a view and saves it as a PNG or BMP file, the base for a static map service.
`06.FMX`	FireMonkey: a `TsgcFMXMap` created in code, with search and routing, location, overlays and view settings on tabs. Delphi XE6.
`07.VectorTiles`	Vector tiles from three sources with three styles. Delphi XE2.
`08.FMXMobile`	FireMonkey: one project for Android, iOS, macOS and Windows, made for phones and tablets, with your position. Delphi 10.4.
`09.OGCServices`	An OpenStreetMap base layer with an overlay from a WMS or WMTS server, chosen from the capabilities of the server. Delphi 2009.
`10.PMTiles`	Maps from PMTiles archives, local or on a web server. Delphi XE2, Registered edition.
`11.ShapefileWKT`	An ESRI shapefile with its DBF attributes over OpenStreetMap, and the overlays of the map imported and exported as Well Known Text.

Demos 06 and 08 are FireMonkey projects; everything else is VCL. Demos that use online tiles or services need an internet connection, and some providers need an API key.

Help topics

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

Topic	What it covers
Introduction	What sgcMaps does and where to start.
Platforms, Editions	The Delphi versions, the target platforms and what each edition includes.
Tiles, layers and providers	Every tile source and provider, with keys, zoom limits and retina tiles.
Markers and clustering, Shapes and labels	The overlays in depth.
Offline tiles, PMTiles, Vector tiles	Base maps without a tile server.
File formats, Printing	GeoJSON, GPX, KML and the other formats, and printing the map.
Demos	One topic per demo, with the minimum Delphi version.
Troubleshooting	More problems and their causes: tiles, API keys, usage policies, cache folders, high DPI screens.
TsgcVCLMap	The reference of the map control, with every property, method and event.

On the website

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

Enjoy putting your data on the map.

---

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