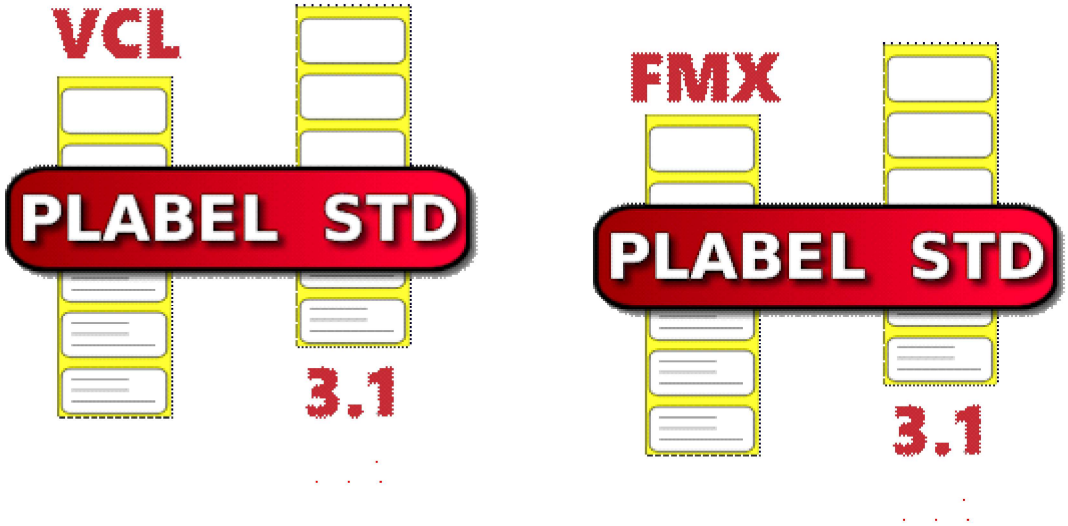
PLABEL VCL & FMX STD



Developer Document

3.1.0

February 2026

1.Page index

2.Introduction

The PLABEL VCL and FMX STD package is a series of components to add label editing and printing to your Delphi **VCL** and **Firemonkey** programs, it's available for 32 and 64 bit versions. The version 3.0, is based on previous versions, but has been rewritten from scratch, now the editor has three modes, apart from editing labels we can design flowcharts and make CAD-type technical drawings, is available for the Seattle, Berlin, Tokyo, Sydney, Rio, Alexandria, Athens and Delphi Florence versions. The most important change to the last version 3.1 is the addition of support for the cross-platform Firemonkey library.

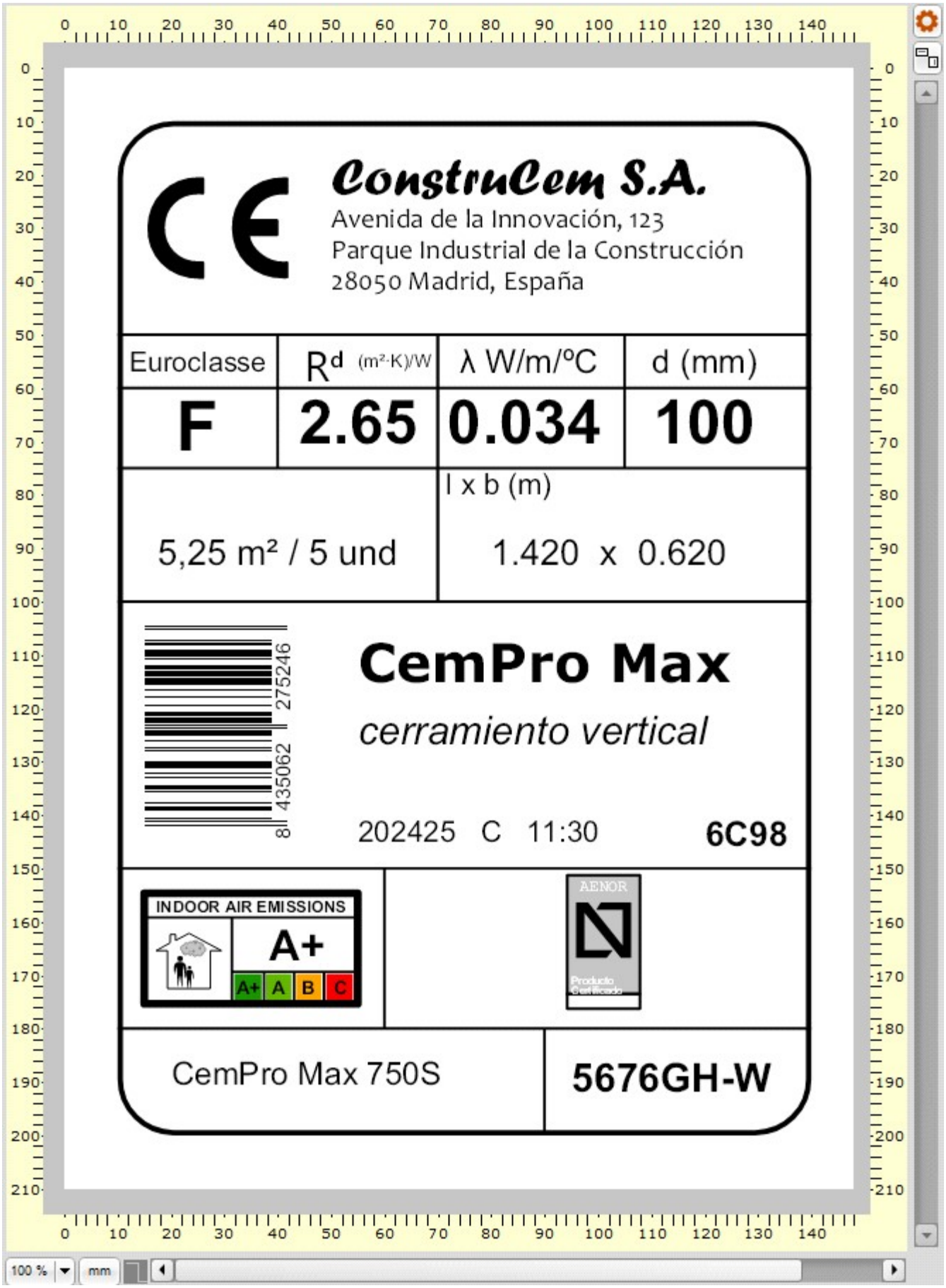


Fig.1 Label editor

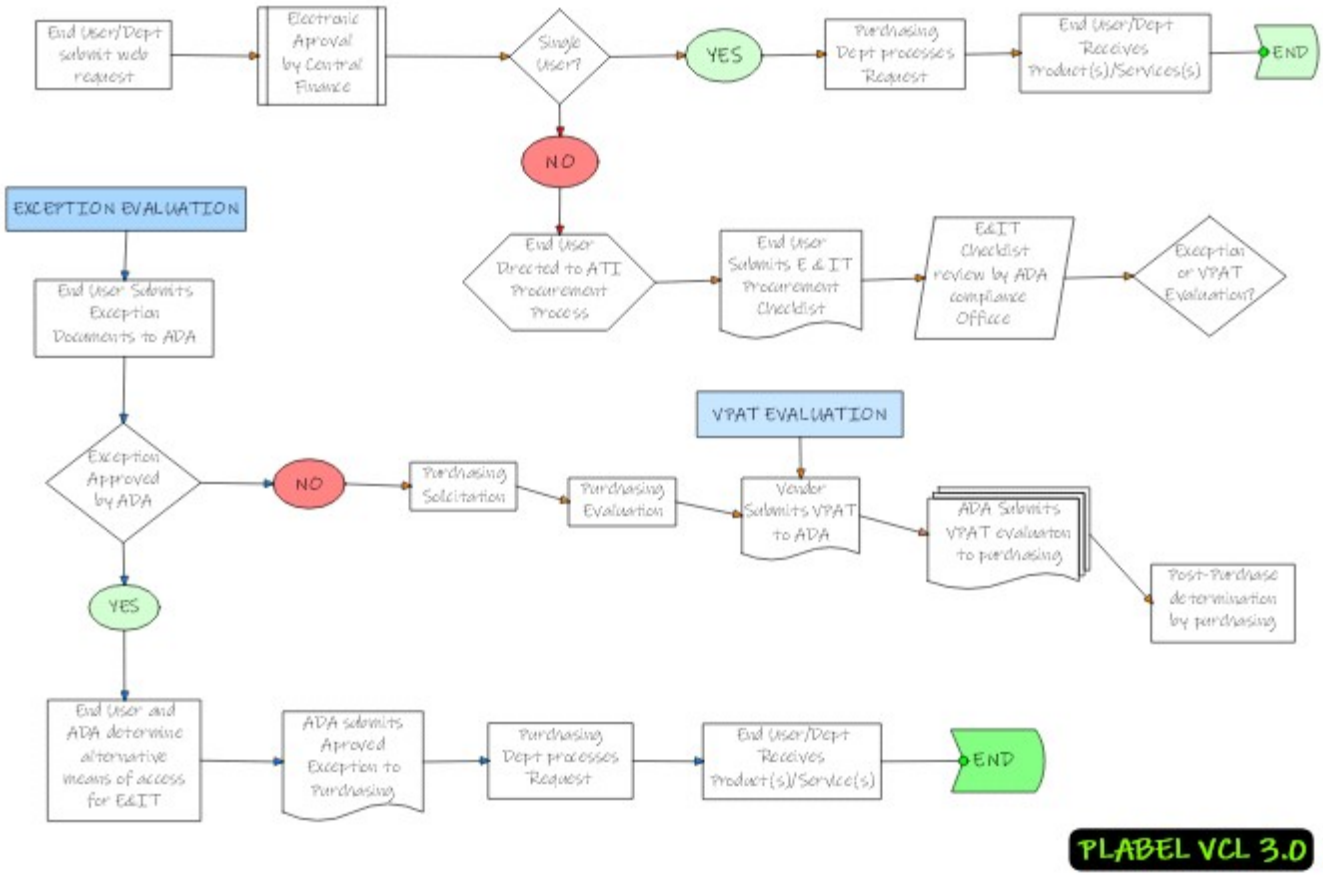


Fig.2 Flow chart

Fig.3 Technical drawing example

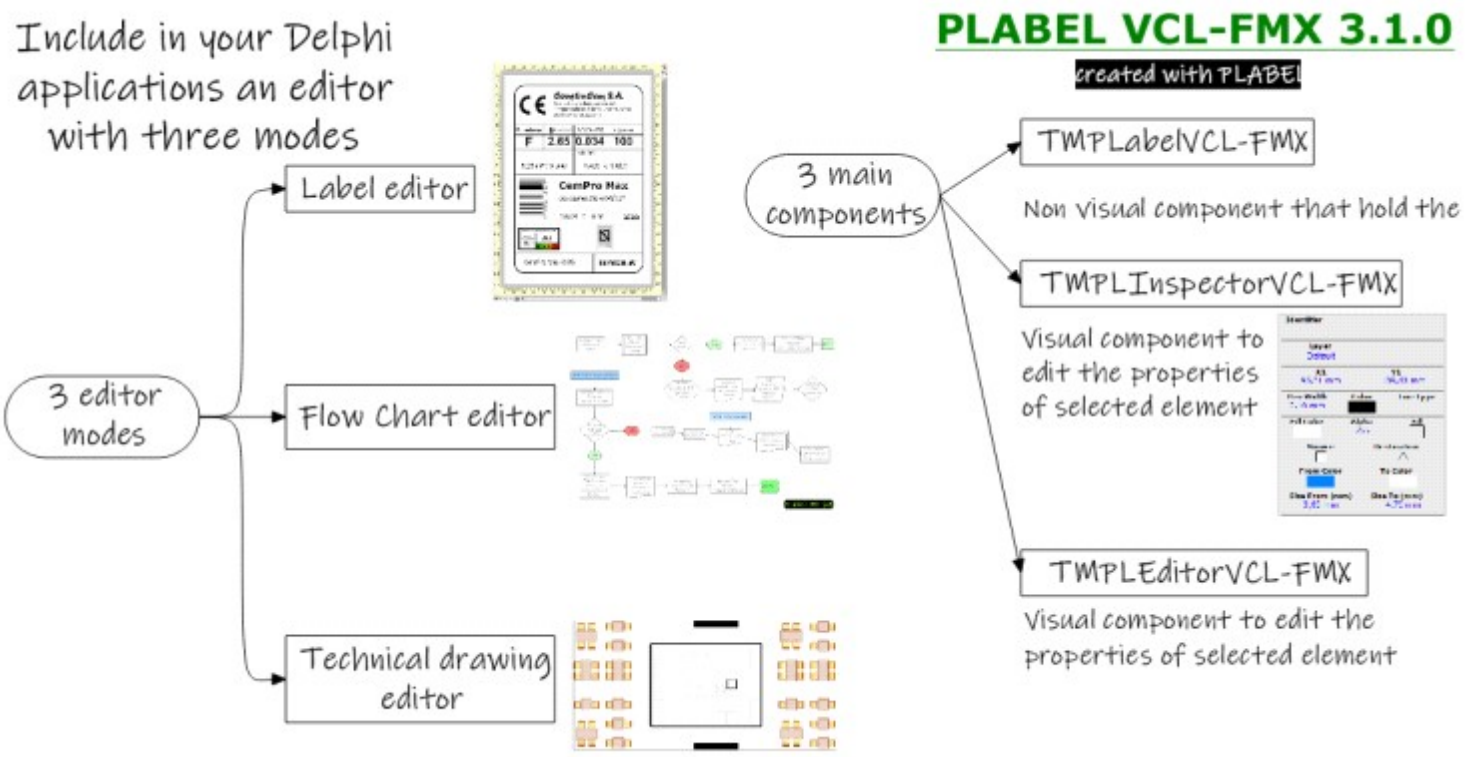


Fig.4 PLABEL VCL-FMX 3.1.0

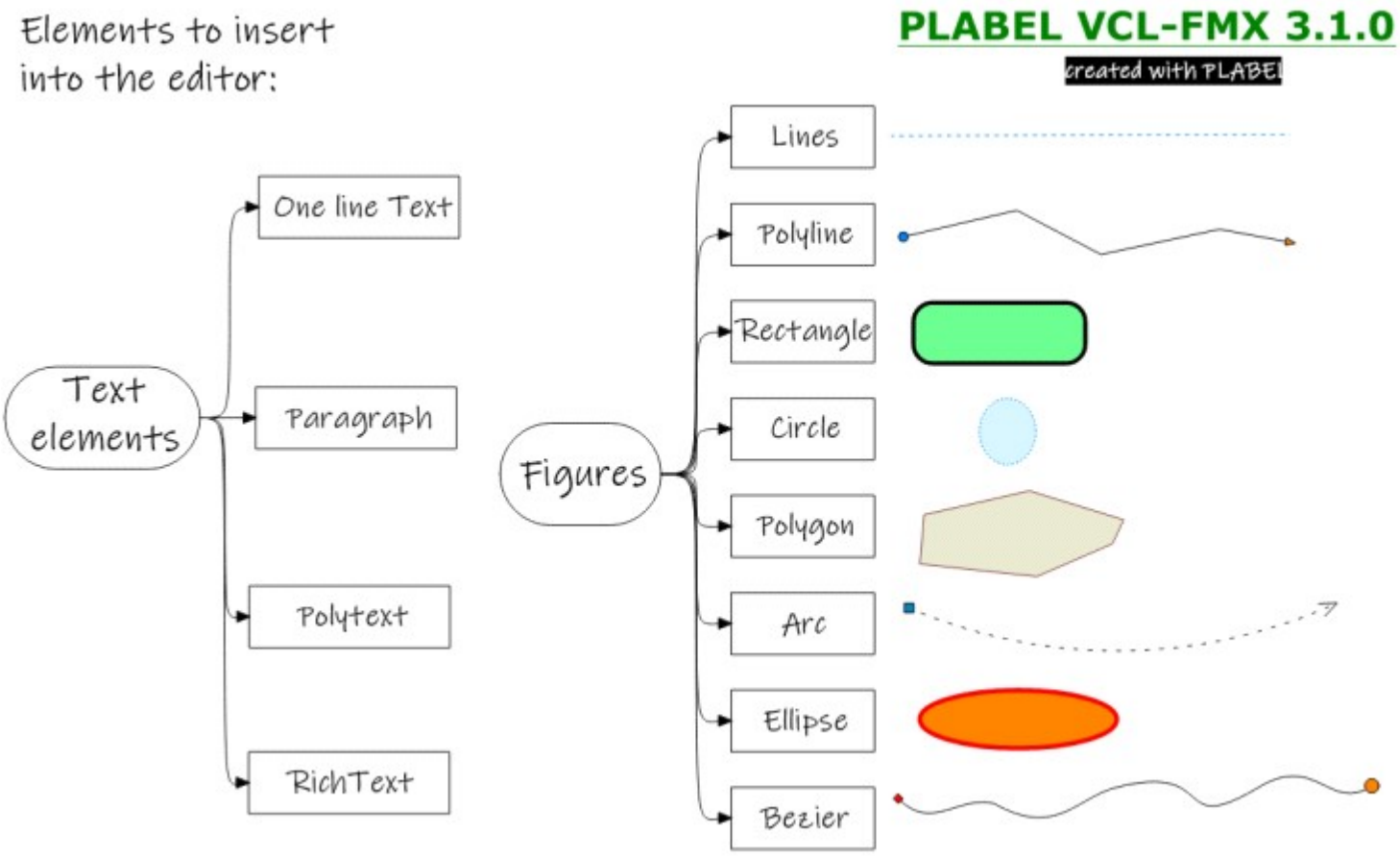


Fig.5 Elements you can insert

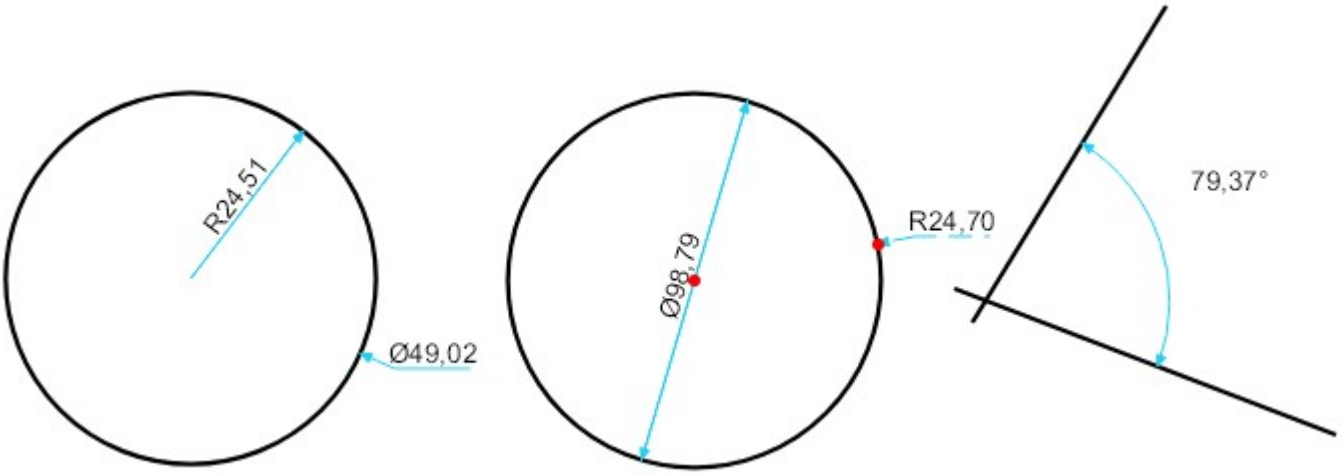


Fig.6 New in version 3.1 radial, diameter and angle dimenion styles.

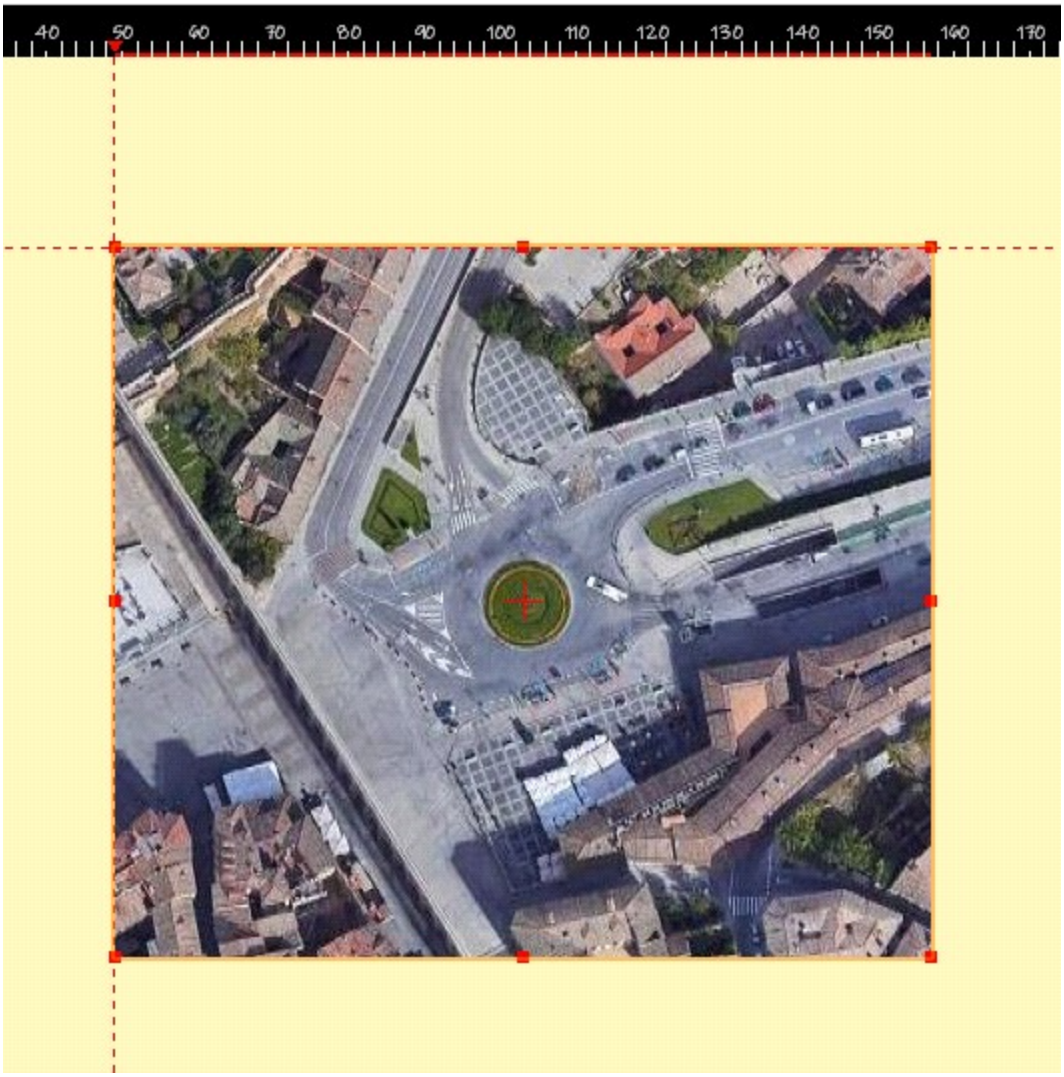


Fig.7 New in version 3.1 Image of map services in the image element or as the background of a layer.

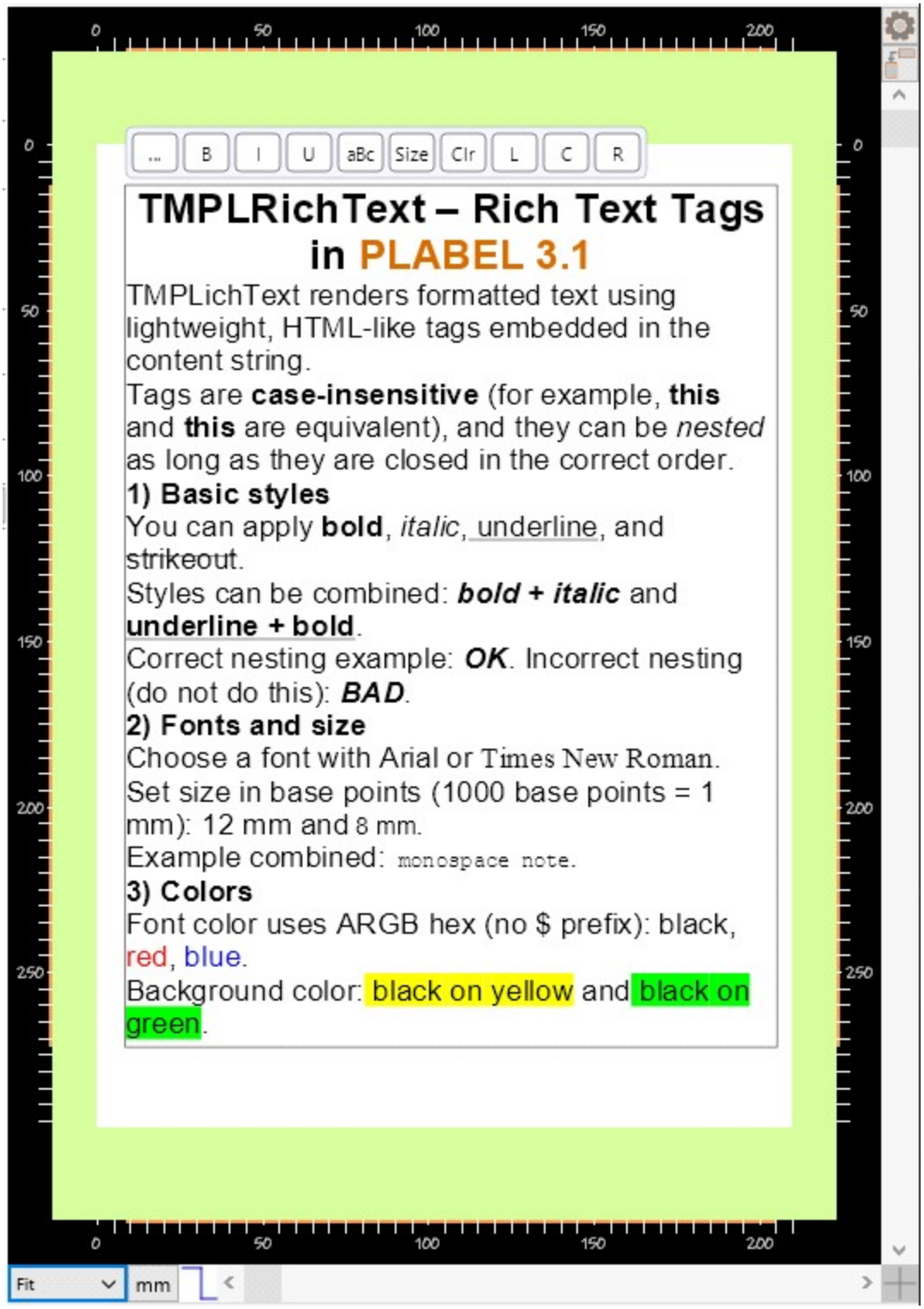


Fig.8 New in version 3.1: TMPLRichText element to show formated text.

These components use the excellent JSON handling library, JsonDataObjects (https://github.com/ahausladen/JsonDataObjects) created by Andreas Hausladen.

3.History

3.1 Version 3.0.0 - August 2024

A version written from scratch of the PLABEL components for VCL. It uses the Skya library to render graphics. New elements and functionalities have been added, such as flowchart and technical drawing modes.

3.2 Version 3.1.0 - January 2026

- The most important change in this version 3.1 is the addition of support for the cross-platform **Firemonkey** library.It also uses the Skia graphics library to render graphics.

- New **TMPLRichText** element that allows you to add various formatting (font size and type, bold, italics, color and alignment) to a text.

- Added is the ability to set an image as the background of a layer, either from a file or from an online map service such as Google Maps or OpenStreetMap, setting its position, zoom and map type. A map image can also be assigned to the image element (TMPLImage).

- We can fix vertical and horizontal alignment lines on the rulers, which can be adjusted with the mouse or by assigning values. Then you can adjust the left, top, bottom, or right element's positions on these guides by calling the corresponding adjustment method.

- New types of snaps as a drawing aid: tangent and perpendicular.

- Operation of extending lines expanded to circle and arc, as reference figures.

- New dimensioning types to indicate radius, diameter, and two-line angle.

- Being able to resize the size that the text occupies in the editor, when it is linked to data, by default we will see the field name as text and sometimes it may be better that it occupies less space in the editor.

- Added DataMatrix 2D barcode to TMPLPolyText element

4.Installation

4.3 Prerequisites

PLABEL VCL uses the Skia library (Skia4Delphi) to display graphics. The Skia Graphics Engine or Skia is an open-source 2D graphics library written in C++. Skia abstracts away platform-specific graphics API (which differ from one to another). Skia Inc. originally developed the library; Google acquired it in 2005, and then released the software as open source licensed under the New BSD free software license in 2008. Embarcadero has followed this excellent project and has incorporated Delphi 12 Athens and Delphi 13 Florence in its latest version. For older versions that use PLABEL, you need to install it manually (although it can be done from GetIt, it is preferable to do so by downloading the installable from the project page to have the most recent version). The installer will ask you which versions of Delphi you want to install it on. This graphics library uses a dll (sk4d.dll) that you must include in your executable folder.

The installation of PLABEL VCL and FMX STD components can be done in two ways:

4.4 Automatic installation

It is the recommended way, run the installation setup PLABEL_VCL_30_ DelphiVersion_RegistrationNumber .exe . Read the license agreement and accept if you agree. The setup will suggest a directory where the packages, code, documentation and examples will be installed. The necessary entries and search paths for the Delphi IDE will be added to the Windows registry.

> The most important thing to note is that you cannot use the components for the sole purpose of designing and printing labels, i.e. the components must be used to incorporate label design and printing in more general use programs.

4.5 Manual installation VCL

If you need to recompile or reinstall the components, go to the folder where the group file is located, the MPL.PALBEL project group has three packages: MPL.PLABEL.RTL_VER (common code for VCL and FMX), MPL.PLABEL.VCL_VER (VCL components) and MPL.PLABEL.DSGN.VCL_VER (component functionalities for design within IDE). VER = Delphi Compiler version.

Open each of the projects in the previous order, on the first run the Build command, on the second run Build and Install (it will install the **TMPLInspector** , **TMPLViewLibrary** , **TMPLEditor** and **TMPLSpinEditFloat** components, and finally when opening the third project run Build and Install which will install the **TMPLabelVCL** component in the PLABEL VCL STD tab.

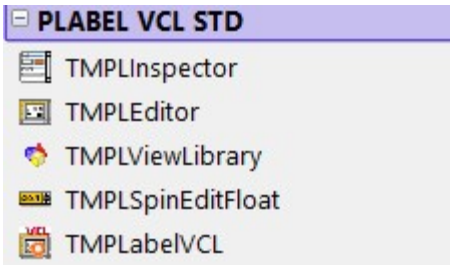


Fig.9 PLABEL VCL components

TMPLSpinEditFloat	Descendant of TCustomEdit that is used in various forms for entering and formatting decimal values.
TMPLEditor	Visual component for designing and editing labels and diagrams, connects to the TMPLabelVCL base component
TMPLInspector	Visual component to view and modify the properties of selected elements in the editor, connects to the TMPLabelVCL base component
TMPLViewLibrary	Used to display a library of *TMPLSymbol* elements in selection dialog.
TMPLabelVCL	Non-visual component that stores all the information of the label or diagram, the connection to the data and configuration options. In some cases we can only use this component in our projects if we want to read, export as file (pdf, svg, png, jpeg and gif), and print files.

Table2 PLABEL VCL components

To reference the folder where the code files are, you have two options:

A)- In each project in Project --> Options --> Delphi Compiler --> Search Path , write, separated by semicolons, the code folders:

- C:\Users\Public\Documents\Mesurasoft\Plabel\VCL\3_1\Source;
- C:\Users\Public\Documents\Mesurasoft\Plabel\VCL\3_1\Source\VCL;
- C:\Users\Public\Documents\Mesurasoft\Plabel\VCL\3_1\Source\Shared

B)- Define a variable (for example *$(PLABEL_VCL)* as in the examples), and in Tools --> Options --> Environment options --> Environment variables add a new variable with the names of the three previous folders separated by commas.

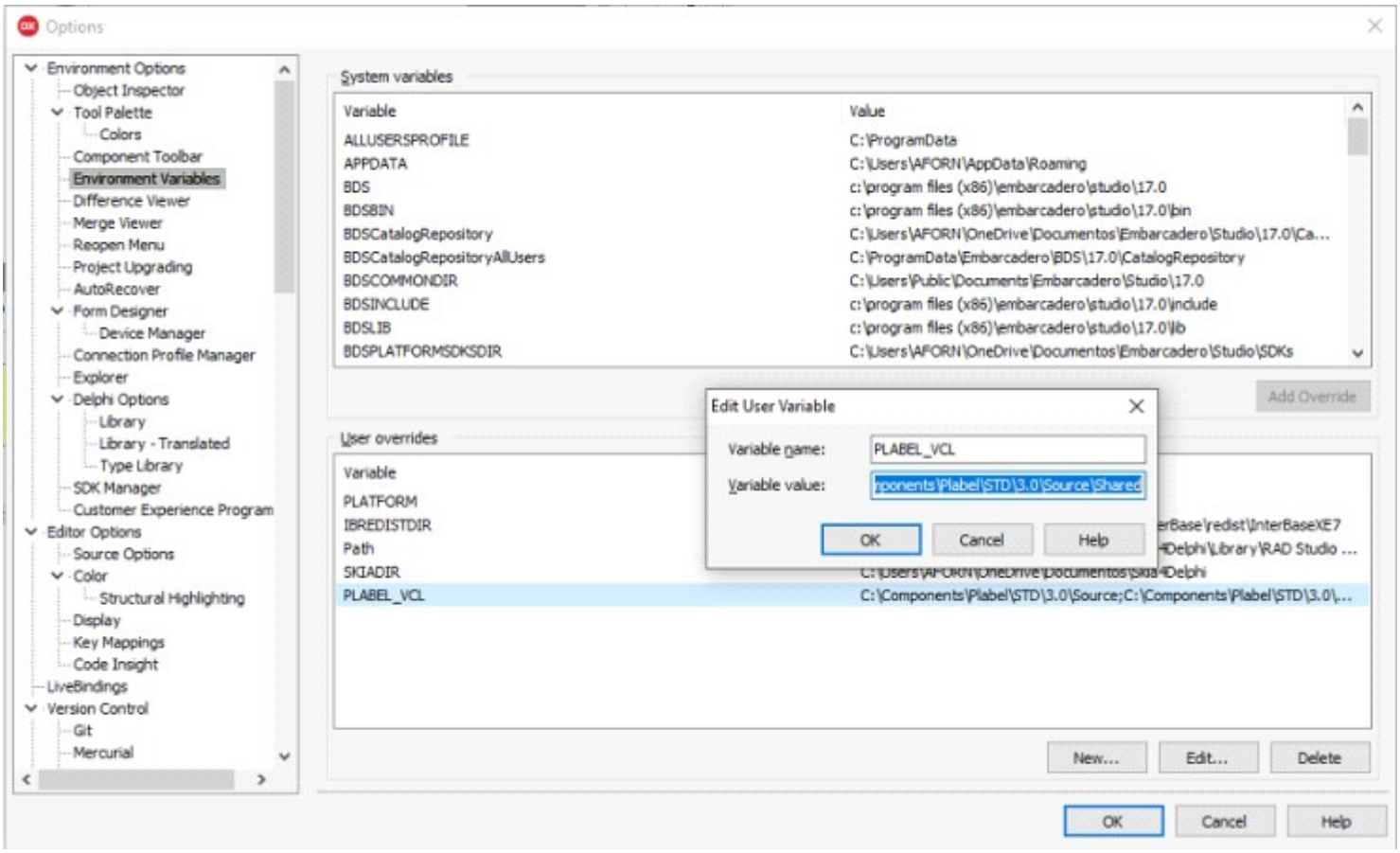


Fig.10 Define new variable

Then in Project options --> Delphi Compiler --> Search Path write the name of this variable *$(PLABEL_VCL)*

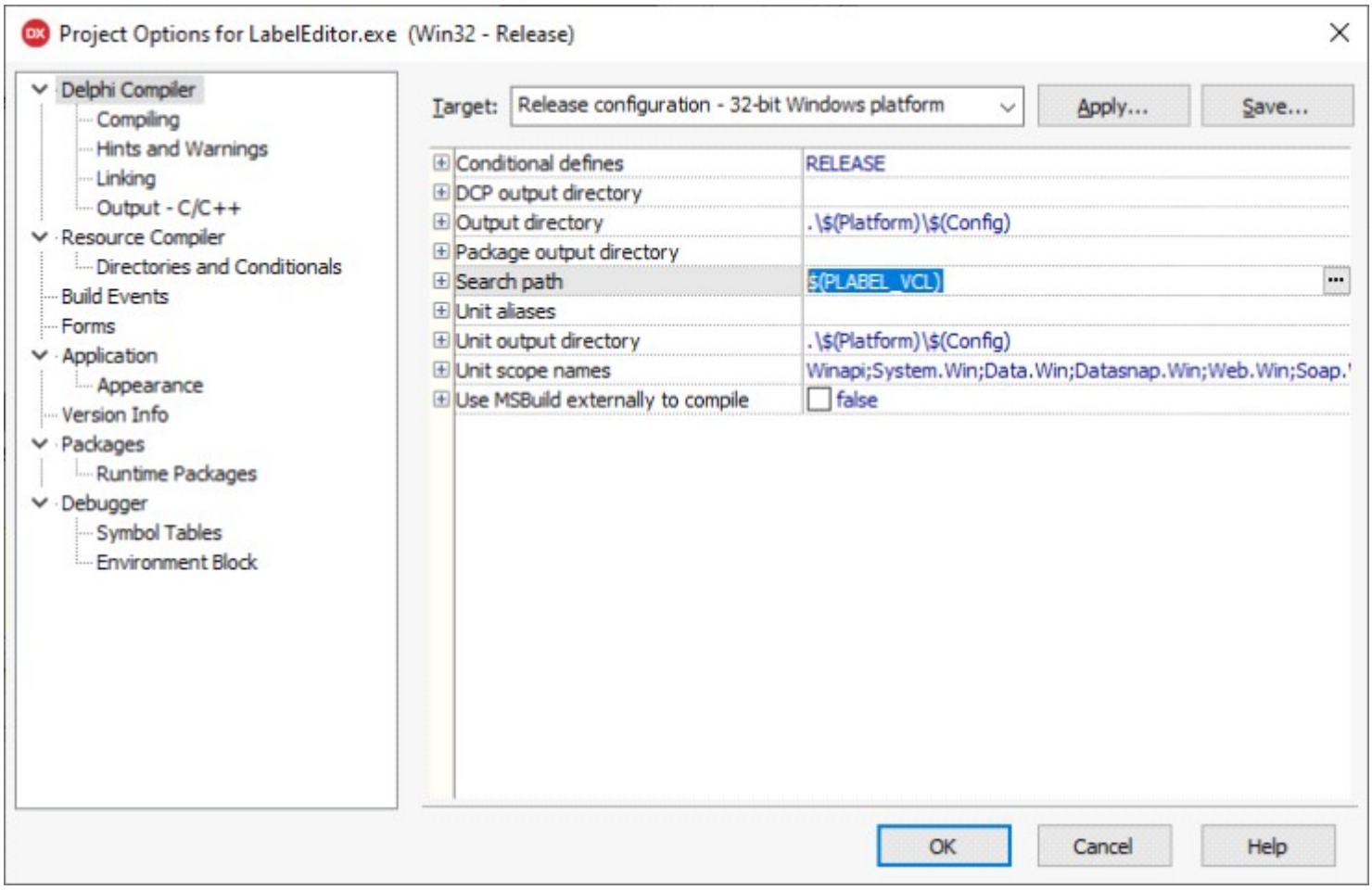


Fig.11 Search path variable

4.6 Manual installation FMX

If you need to recompile or reinstall the components, go to the folder where the group file is located, the MPL.PALBEL project group has three packages: MPL.PLABEL.RTL_VER (common code for VCL and FMX), MPL.PLABEL.FMX_VER (FMX components) and MPL.PLABEL.DSGN.FMX_VER (component functionalities for design within IDE). VER = Delphi Compiler version.

Open each of the projects in the previous order, on the first run the Build command, on the second run Build and Install (it will install the **TMPLInspector** , **TMPLViewLibrary** and **TMPLEditor** components, and finally when opening the third project run Build and Install which will install the **TMPLabelFMX** component in the PLABEL FMX STD tab.

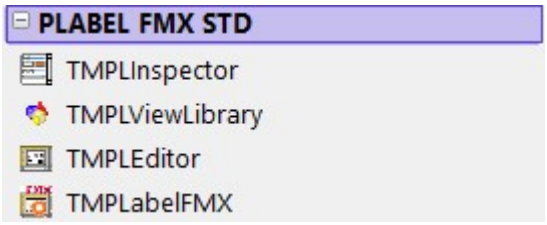


Fig.12 PLABEL FMX Components

TMPLEditor	Visual component for designing and editing labels and diagrams, connects to the TMPLabelFMX base component

TMPLInspector	Visual component to view and modify the properties of selected elements in the editor, connects to the TMPLabelFMX base component
TMPLViewLibrary	Used to display a library of *TMPLSymbol* elements in selection dialog.
TMPLabelFMX	Non-visual component that stores all the information of the label or diagram, the connection to the data and configuration options. In some cases we can only use this component in our projects if we want to read, export as file (pdf, svg, png, jpeg and gif), and print files.

Table [n]. PLABEL FMX components

To reference the folder where the code files are, you have two options:

A)- In each project in Project --> Options --> Delphi Compiler --> Search Path , write, separated by semicolons, the code folders:

- C:\Users\Public\Documents\Mesurasoft\Plabel\FMX\3_1\Source;
- C:\Users\Public\Documents\Mesurasoft\Plabel\FMX\3_1\Source\FMX;
- C:\Users\Public\Documents\Mesurasoft\Plabel\FMX\3_1\Source\FMX\DelphiVersion
- C:\Users\Public\Documents\Mesurasoft\Plabel\FMX\3_1\Source\Shared

B)- Define a variable (for example *$(PLABEL_FMX)* as in the examples), and in Tools --> Options --> Environment options --> Environment variables add a new variable with the names of the three previous folders separated by commas.

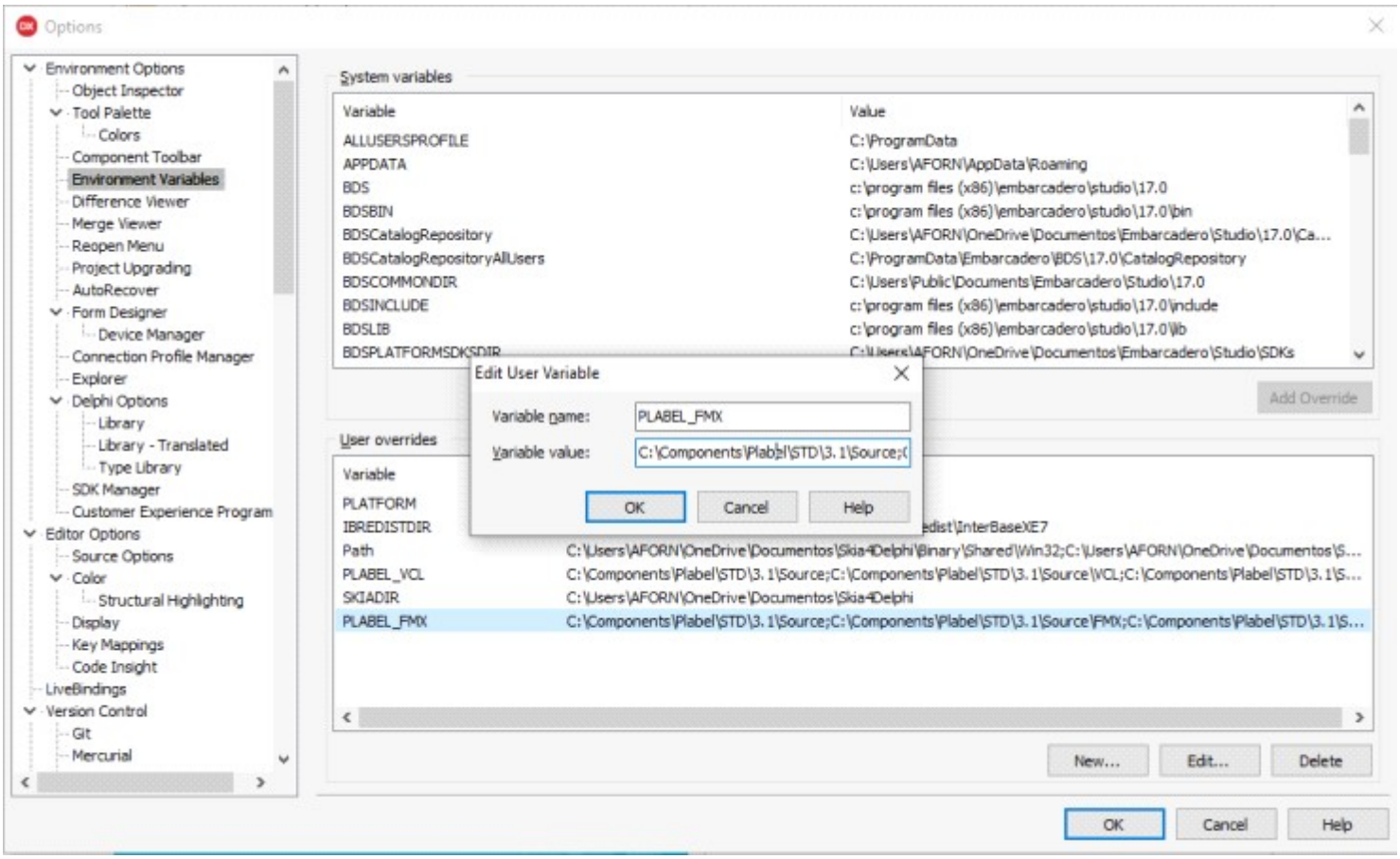


Fig.13 Define FMX variable paths

Then in Project options --> Delphi Compiler --> Search Path write the name of this variable *$(PLABEL_FMX)*

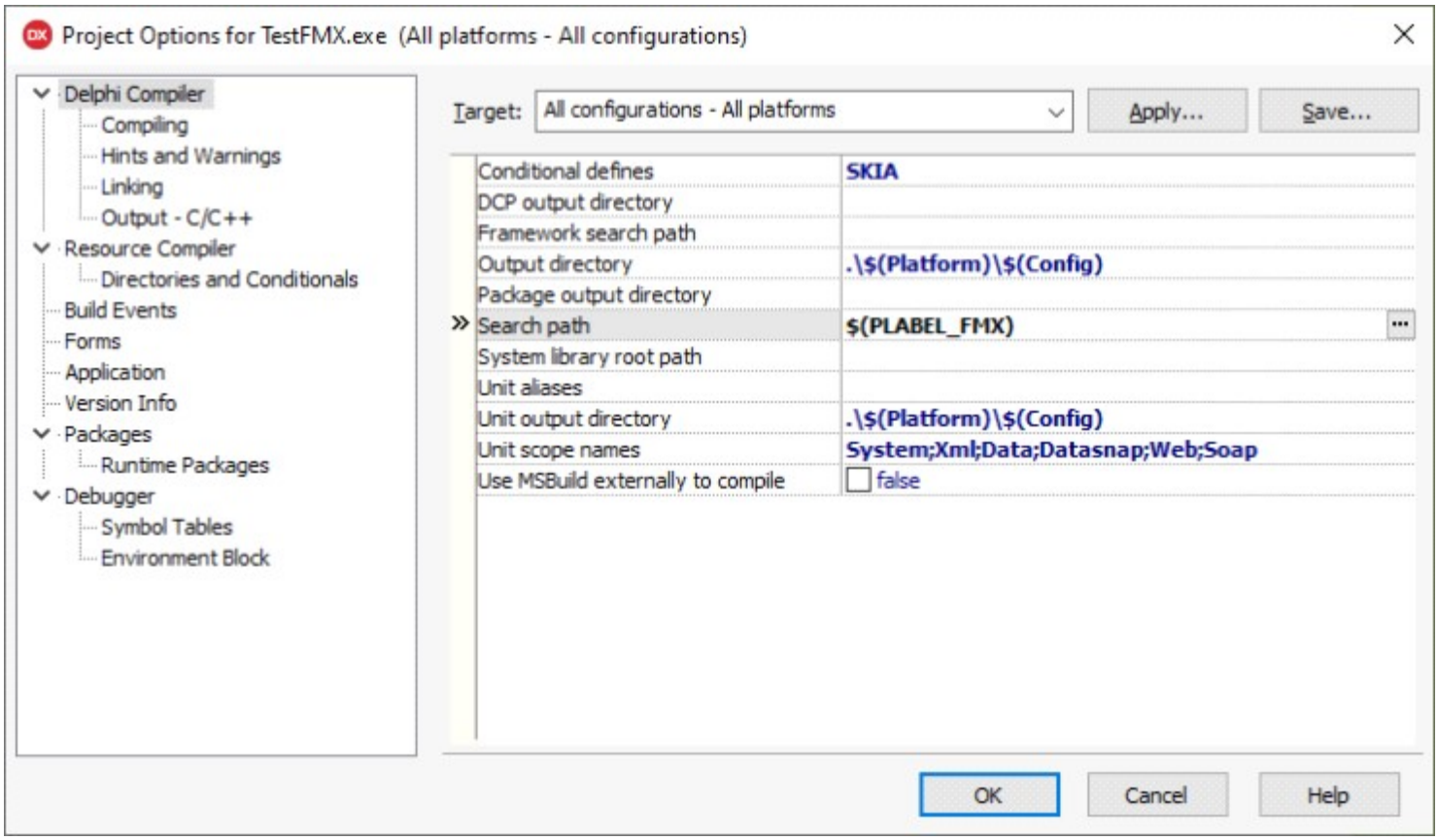


Fig.14 Assign the search path variable

5.Internationalization

The **PLABEL VCL STD** components allow you to define the language used in your forms and messages. This translation into the desired language is done by reading an ini file that we will assign to the *LanguageFileName* property of the **TMPLabelVCL** and **TMPLabelFMX** components. If we do not specify a file, the default language will be English.

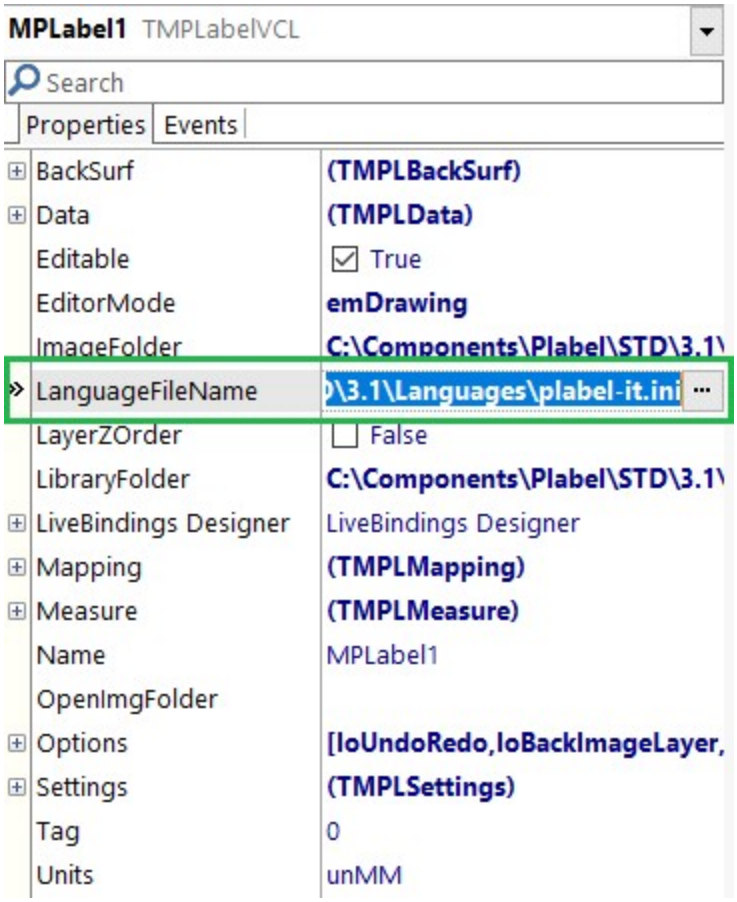


Fig.15 LanguageFileName property

If you cannot find your language in the provided Languages folder's ini files, you can make a copy of plabel-en.ini and translate it to your language. You can rename the name key in the language section above:

```
[language]
name=english
```

and save the file with a language identifier that suits your needs.

6.Components

6.7.TMPLabelVCL - TMPLabelFMX

6.7.1 Properties

Action

With this property we assign the type of action (insertion of elements, or editing operations) active in the editor. Once the action is executed, we can execute it again with the right mouse button in the editor.

```
procedure  TFMain.butPolylineClick (Sender: TObject);
begin
 MPLabel.Action := eaPolyline;
end;
```

The actions to **insert elements** are: eaPoint, eaLine, eaArrowLine, eaRectangle, eaPolyline, eaArc, eaPolygon, eaBezier, eaCirclePR, eaCircle2P, eaCircle3P, eaEllipse, eaText, eaParagraph, eaRichText, eaBarcode, eaPDF417, eaDataMatrix, eaQRCode, eaImage, eaFlow, eaLink, eaPolyLink, eaTable, eaPolyText, eaRank, eaNutriScore, eaLevel, eaSymbol, eaDim, eaDimRad1, eaDimRad2, eaDimDia1, eaDimDia2, eaDimAng, eaSectorPR, eaSector2P, eaSector3P, eaPackage.

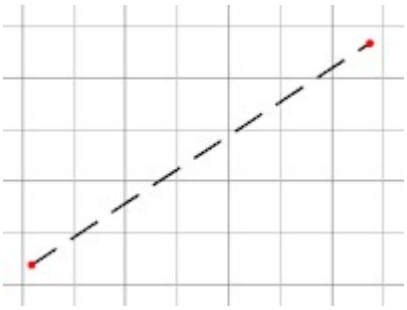
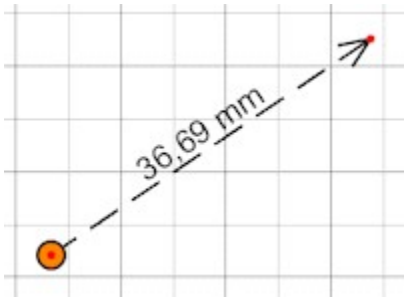
eaLine	eaArrowLine
	

Table3 Line elements I

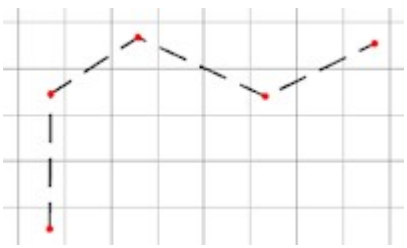
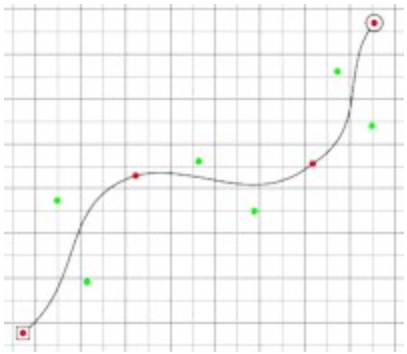
eaPolyline	eaBezier
	

Table4 Line elements II

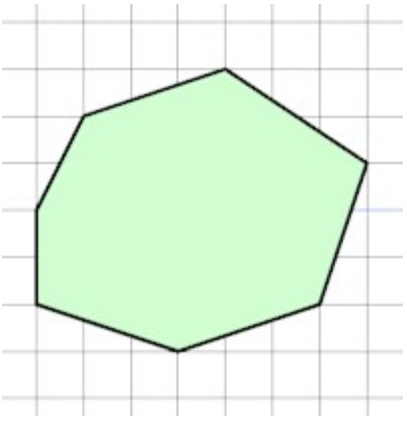
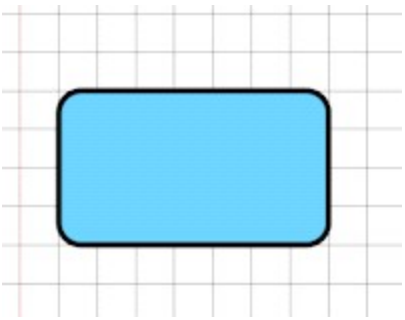
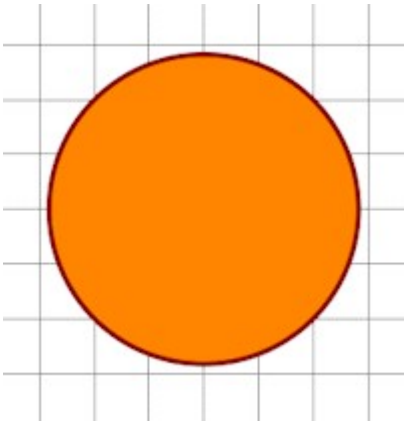
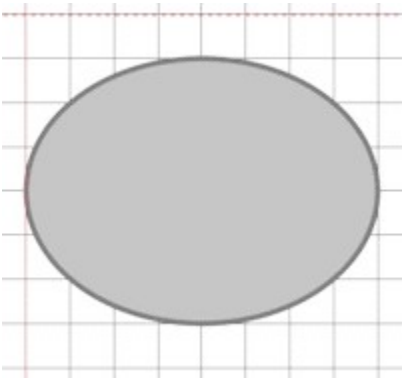
eaPolygon	eaRectangle
	
esCircle (point radio, two points and 3P)	eaEllipse
	

Table5 2D Elements I

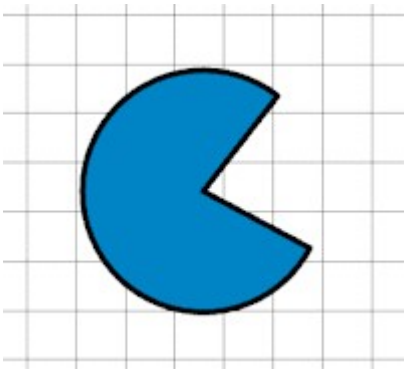
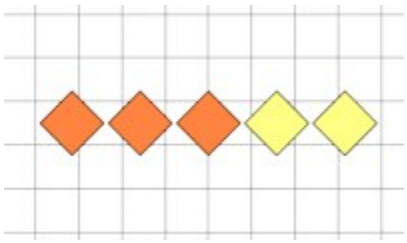
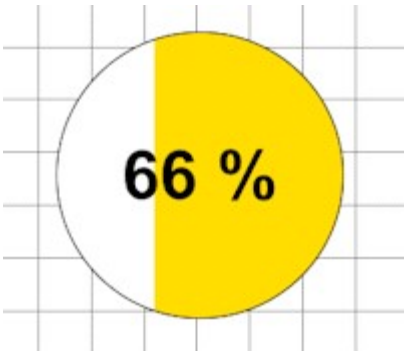
eaSector	eaRank
	

Table6 2D Elements II

eaLevel	eaImage



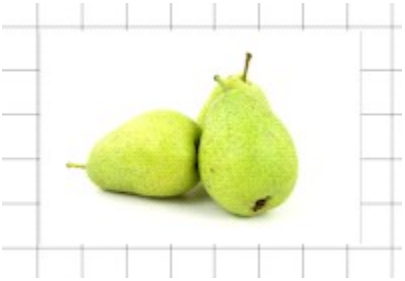
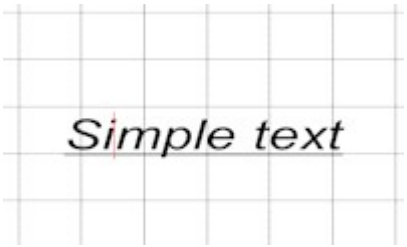


Table7 2D Elements III

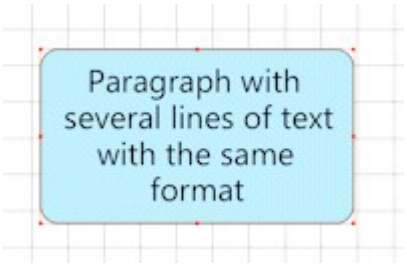
eaText



eaPolytext

Table8 Text elements I

eaParagraph



eaRichText

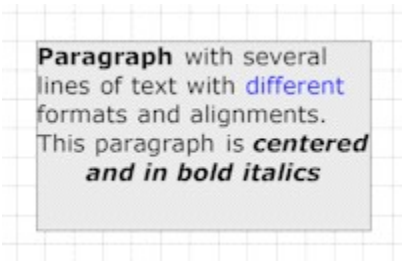


Table9 Text elements II

eaNutriScore

eaSymbol

Table10 Symbolism

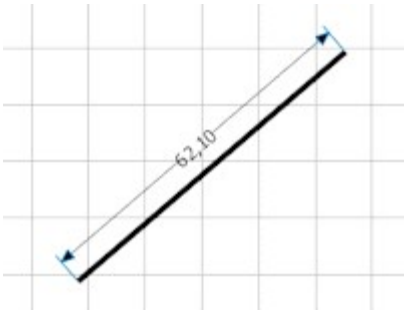
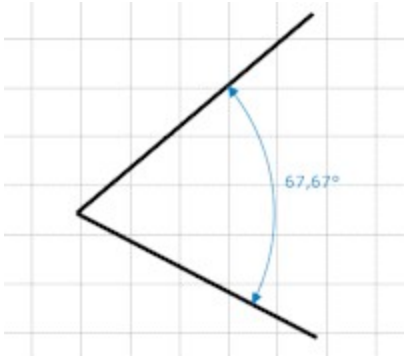
eaDim	eaDimAng
	

Table11 Dimension types I

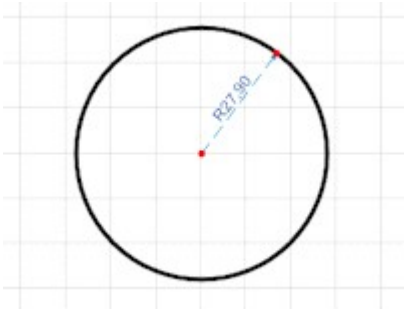
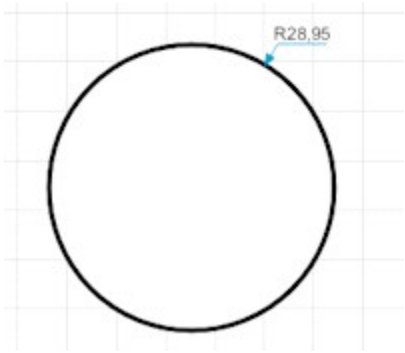
eaDimRad1	eaDimRad2
	

Table12 Dimension types II

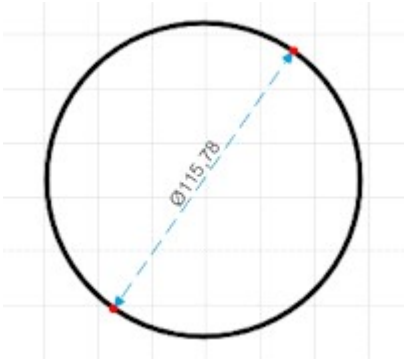
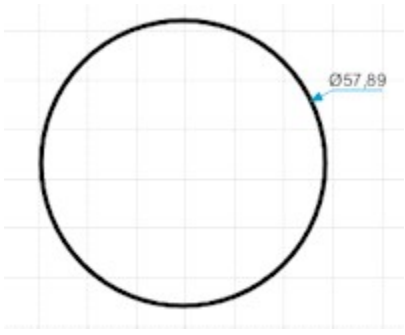
eaDimDia1	eaDimDia2
	

Table13 Dimension types III

eaBarcode (1D)	eaQRCode
	

Table14 Barcodes I

eaDataMatrix	eaPDF417
	

Table15 Barcode II

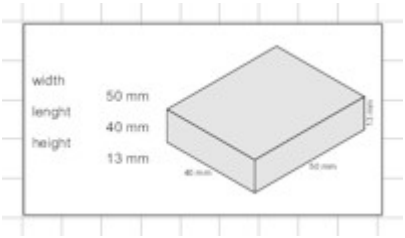
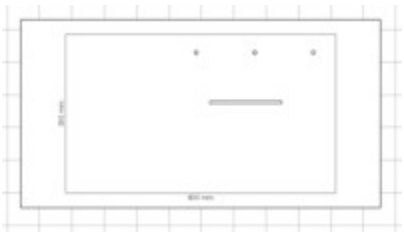
eaPackage (package 2D or 3D dimension)	eaPackage (2D part with drills)
	

Table16 Package

eaTable

variedad	categoria	cajas	confección
Melocotón	I	40	Caja de cartón 5 Kg Alveolo 48
Nectarina	II	30	Bandeja plástica 2kg
		70	

variedad	categoria	cajas	confección
Albaricoque	Extra	25	Caja de madera 8 kg granel
Ciruela	I	35	Caja de cartón 6 kg alveolo 32
Paraguayo	II	20	Bandeja plástica film 800 gr
		80	

Table17 Table element

The actions to **edit operations** are: eaSelection, eaTrim, eaJoin, eaExtend, eaEquidist.

BackSurf.Color

Type TAlphaColor. Color used as the label background in the editor.

Data

Used in editor's Labelling mode. It is the label's data source configuration layer: project datasets, JSON, CSV, value lists, and pre-print/preview questions. At design time you can associate Delphi components, descendants of TDataSet, which the user can choose in the label properties dialog box. At design time, you can associate Delphi components, descendants of TDataSet, which the user can select in the label properties dialog box. We can define the text used to identify the TDataSet and its fields, as well as how to identify them. A TDataSet can have one or more linked TDataSets in the form of a Detail, used to display his data in TMPLTable element.

DrawScale

DrawScale represents the drawing scale ratio 1:N. A value of 50 means the drawing is at scale 1:50—one millimeter on paper represents 50 millimeters in reality. The 1:N scale multiplier, only active in emDrawing and emInteractive (forced to 1 in other modes). Includes the coordinate flow formulas. Extension lines, gaps, arrows: style values are multiplied by DrawScale so they appear the same physical size on paper regardless of scale.

EditorMode

With this property we establish the three available editing modes: *emLabelling* , *emDrawing* and *emFlowChart* . Each mode affects different aspects such as rotating right or left (only available in emLabelling), or applying a scale other than 1 (emDrawing).

Editable

Type Boolean. Property of the editor state where we indicate whether we allow modifying, moving or adding new elements.

ImageFolder

We specify the folder where the images that we will insert into the TMPLImage element or Layers background image will be copied.

LanguageFileName

We specify the ini file with the translation to the language we want to display in the dialog boxes, properties and messages.

LayerZOrder

Boolean property in TMPLabelVCL and TMPLabelFMX to control whether items are automatically sorted according to the layer order.

- When it is true, the component reorders fItems by layer index (SortItemsZLayer method).
- When it is set to false, this automatic layer order is not applied.
- It affects the internal order of drawing/printing/exporting based on the list of elements.

LibraryFolder

When we want to insert a TMPLSymbol element, a dialog box opens that will display the symbols from the various *.lyp files we have in this folder. (See below for how to create these symbol files)

Mapping

Use this property to configure geographic map composition (provider, center, zoom, API key, and provider list), to generate a map image that can be assigned as layer background or to a TMPLimage element.

- Define map center using Latitude / Longitude.
- Select map source with Provider.
- Limit selectable providers in UI dialogs with MapProviders.
- Set zoom level with Zoom (0..20).
- Pass provider API key through MapApi.
- Build tile requests and compose the final map image via LoadMap and component ComposeImage. Normally this will be called with the build in dialog. In this dialog you can call a Localization service to find Latitude / Longitude values (Search button). After you set the coordinates, you push Request Map button and wait until OK button is activated. If you check Show Map center, map composition can draw a center marker.
- Optionally apply computed map scale to editor drawing scale with ApplyScale. (If you check Assign Drawing Scale).

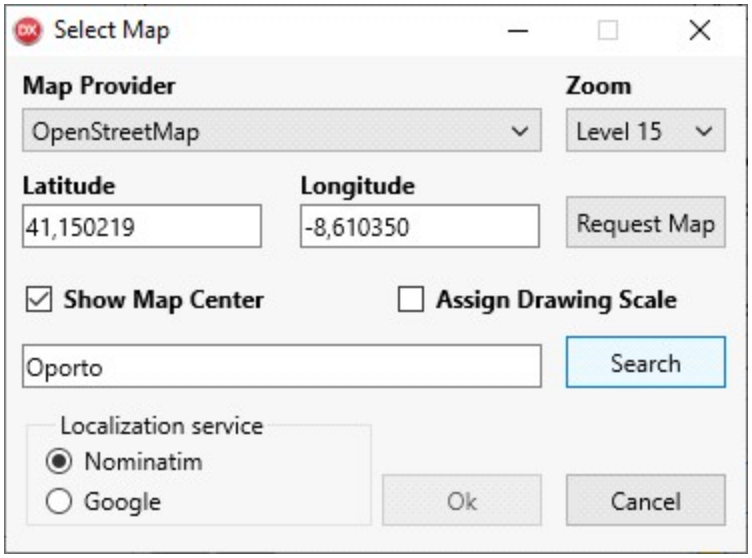


Fig.16 Select Map dialog (Layers editor)

Before assigning a background image to a layer, or to a TMPLImage item, you must specify the folder where the selected images will be copied.

Measure

Controls label geometry, layout grid, paper size, and drawing metadata.

Property	Type	Description
Rows, Columns	integer	Grid count for labels on the paper. Minimum enforced is 1.
Width, Height	single	Label size. Minimum enforced is 5 mm. In circle mode, width and height are kept equal.
MarginLeft, MarginTop	single	Top-left offsets for label placement. Minimum allowed is -5 mm.
GapRows, GapColumns	single	Spacing between labels. Negative values are clamped to 0.
PaperWidth, PaperHeight	single	Paper size. Minimum enforced is 5 mm. Can auto-sync with label size in continuous/drawing modes.
PaperName	string	Named paper format (for example A4). Updating this can also update paper dimensions.
ContinousPaper	boolean	When true, forces single label layout (1x1), zero gaps/margins, and label size equals paper size. (normally used in industrial printers).
DrawScale	integer	Drawing scale used in drawing /interactive mode editors. Forced to 1 outside those modes.
LabelName, Promoter, Author, Project, Title, Date, Note, PlanNumber	mixed	Descriptive metadata for layouts and plans with drawing mode.

Format, IncludeUnit	string, boolean	Measurement text formatting and whether units are included.

Table18 Measure property

- SetLabelSizeMM and SetPaperSizeMM are helper methods that assign values directly in millimeters.
- In editor modes emFlowChart, emDrawing, and emInteractive, continuous-paper behavior is effectively forced.

```
// Same for TMPLabelVCL and TMPLabelFMX
MPLabel1.Measure.PaperName := 'A4';
MPLabel1.Measure.Rows := 7;
MPLabel1.Measure.Columns := 3;
MPLabel1.Measure.Width := 60.0;
MPLabel1.Measure.Height := 35.0;
MPLabel1.Measure.MarginLeft := 5.0;
MPLabel1.Measure.MarginTop := 10.0;

// Direct size assignment in mm
MPLabel1.Measure.SetLabelSizeMM(60.0, 35.0);
MPLabel1.Measure.SetPaperSizeMM(210.0, 297.0);
```

OpenImgFolder

When we open the dialog box to select an image, we indicate here the folder that will open by default.

Options

Option	Default	Purpose
loUniqueID	ON	Enforces unique ID values across items — raises an exception if a duplicate is assigned
loUndoRedo	ON	Enables the undo/redo command stack (TMPLCmd). Automatically suspended during file loading
loBackImageLayer	OFF	Shows the "Background Image" menu in the Layers dialog, allowing static image backgrounds per layer
loBackImageMap	OFF	Shows the "Background Map" menu in the Layers dialog, allowing map-based backgrounds per layer (typically for Drawing mode)

Table19 Options property

Settings

Defines global visual settings for flow-link terminators and spacing, used in emFlowChart editor's mode. You can change this properties from editor's settings dialog.

Property	Type	Description	Default

TermSize	single	Terminator (arrow/head) size in component units. Internally stored in base mm (thousandths of mm). Effective minimum: 0.5 mm.	2.5 mm
LinkGap	single	Gap used by links/flow elements. Internally stored in base mm. Negative assigned values are clamped to 0.	5.0 mm
TermColor	TAlphaColor	Terminator stroke color.	TAlphaColorRec.Black
FillTermColor	TAlphaColor	Terminator fill color.	TAlphaColorRec.White

Table20 Settings property

Units

Measure units used, it depends of editor modes: Labelling/FlowChart only accept unMM, unINCH, unCM (larger units fall back to unMM). Drawing/Interactive accept all units. TMPLUnits = (unMM, unINCH, unCM, unDM, unM, unFt, unYd);

6.7.2 Methods

AddLink(const Id1, Id2: string; Idx1, Idx2: smallint; aType: TMPLinkType)

Creates a link item between two identifiers/indices through the factory and returns the created link.

BeginUpdate

When you execute several drawing operations in the editor by code, you have to start by calling the **BeginUpdate** method, then we write the code to create elements or modify properties, and we finish with an **EndUpdate** , with this we manage to execute as a whole and optimize the drawing.

```
MPLabel.BeginUpdate;
MPLabel.EditorMode := emDrawing;
MPLabel.Units := unCM;
MPLabel.Measure.DrawScale := 10;
MPLabel.Measure.SetPaperSizeMM(210,297);

with TMPLFactory.AddRectangle(MPLabel,40,40,120,180) do
 begin
  Color := TAlphaColorRec.Chocolate;
  PenWidth := 5;
  Radio := 0;
  Filled := false;
 end;

with TMPLFactory.AddRectangle(MPLabel,47,47,106,166) do
 begin
  Color := TAlphaColorRec.Burlywood;
  BackColor := TAlphaColorRec.Aliceblue;
  PenWidth := 2;
  Radio := 0;
 end;

with TMPLFactory.AddLine(MPLabel,100,47,100,213) do
 begin
```

```
   Color := TAlphaColorRec .Burlywood ;
   PenWidth := 2;
  end;

 with TMPLFactory .AddLine (MPLabel ,47,102,153,102 ) do
  begin
   Color := TAlphaColorRec .Burlywood ;
   PenWidth := 2;
  end;

 with TMPLFactory .AddLine (MPLabel ,47,158,153,158 ) do
  begin
   Color := TAlphaColorRec .Burlywood ;
   PenWidth := 2;
  end;

 with TMPLFactory .AddArrowLine (MPLabel ,40,30,160,30 ) do
  begin
   FontName := 'Ink Free';
   TextPos := tpMiddleRemove ;
   TextMeasure := true;
   FromType := ttFillArrow ;
   FromColor := TAlphaColorRec .Chocolate ;
   ToType := ttFillArrow ;
   ToColor := TAlphaColorRec .Chocolate ;
   Color := TAlphaColorRec .Crimson ;
   FontHeight := 5;
  end;

 with TMPLFactory .AddArrowLine (MPLabel ,170,40,170,220 ) do
  begin
   FontName := 'Ink Free';
   TextPos := tpMiddleRemove ;
   TextMeasure := true;
   FromType := ttFillArrow ;
   FromColor := TAlphaColorRec .Chocolate ;
   ToType := ttFillArrow ;
   ToColor := TAlphaColorRec .Chocolate ;
   Color := TAlphaColorRec .Crimson ;
   FontHeight := 5;
  end;

 MPLabel .Measure .Title := 'Ventanales' ;
 MPLabel .EndUpdate ;
```

(*) Use the methods of **TMPLFactory** class (*MPL.Shared.Factory* unit) to add new elements to the editor, as the previous code.

BringToFront

We place the selected element above the rest, so that it will be drawn last and will be the first to be selected.

EditDimStyles

In technical drawing mode, we open the dimensioning style editor. Editing styles are applied to the dimensions that have them assigned, from the properties inspector we can change the style of a

dimension.

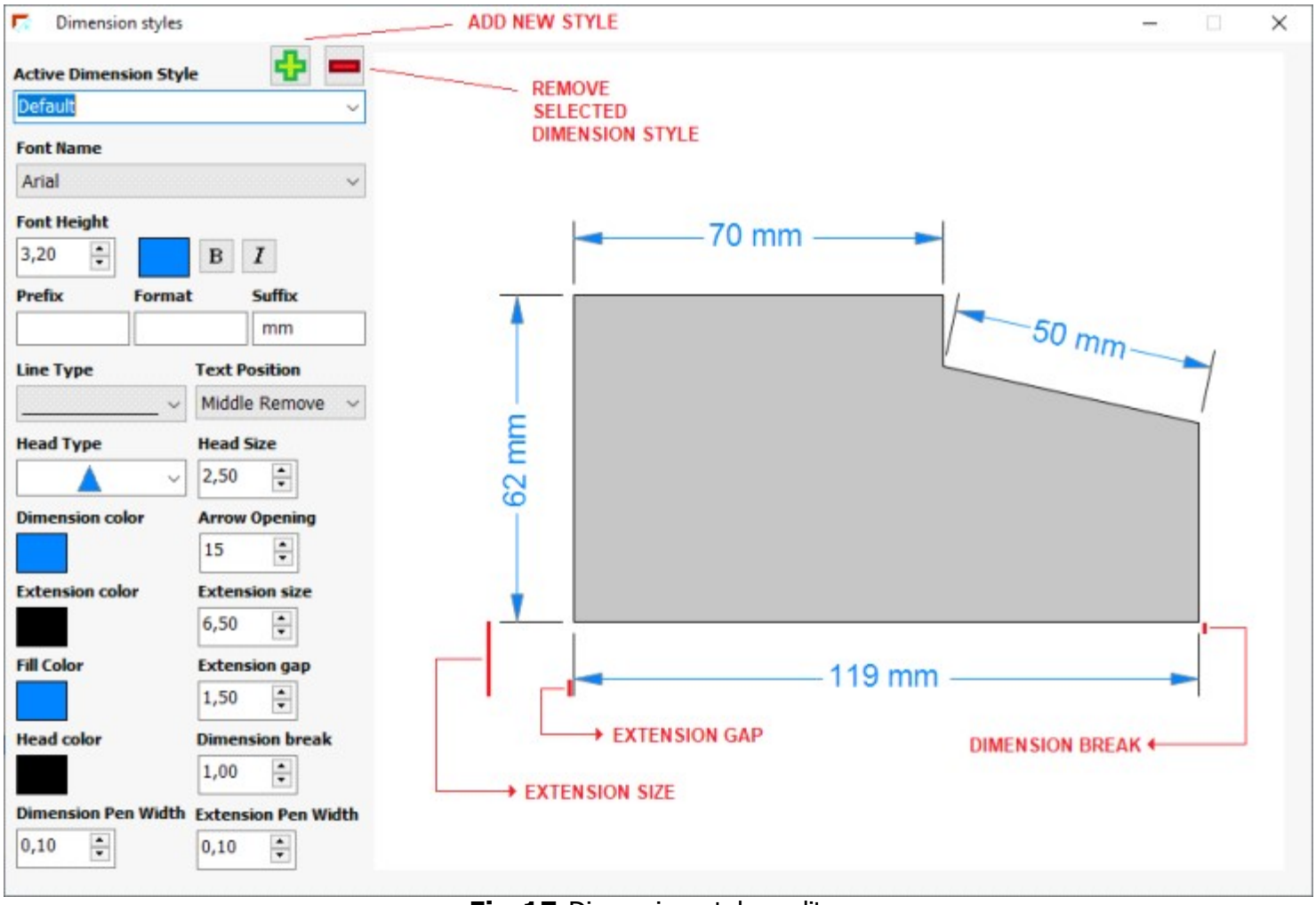


Fig.17 Dimension styles editor

EditLayers

With this action we open the layer editor, where we can add and change their properties, assign as active layer (double click), which is the layer where the elements we add are assigned by default. We can make a layer visible or not, mark it for export (pdf or image), for printing or not or for it to be selectable or not. By default, a layer is created and it cannot be deleted.

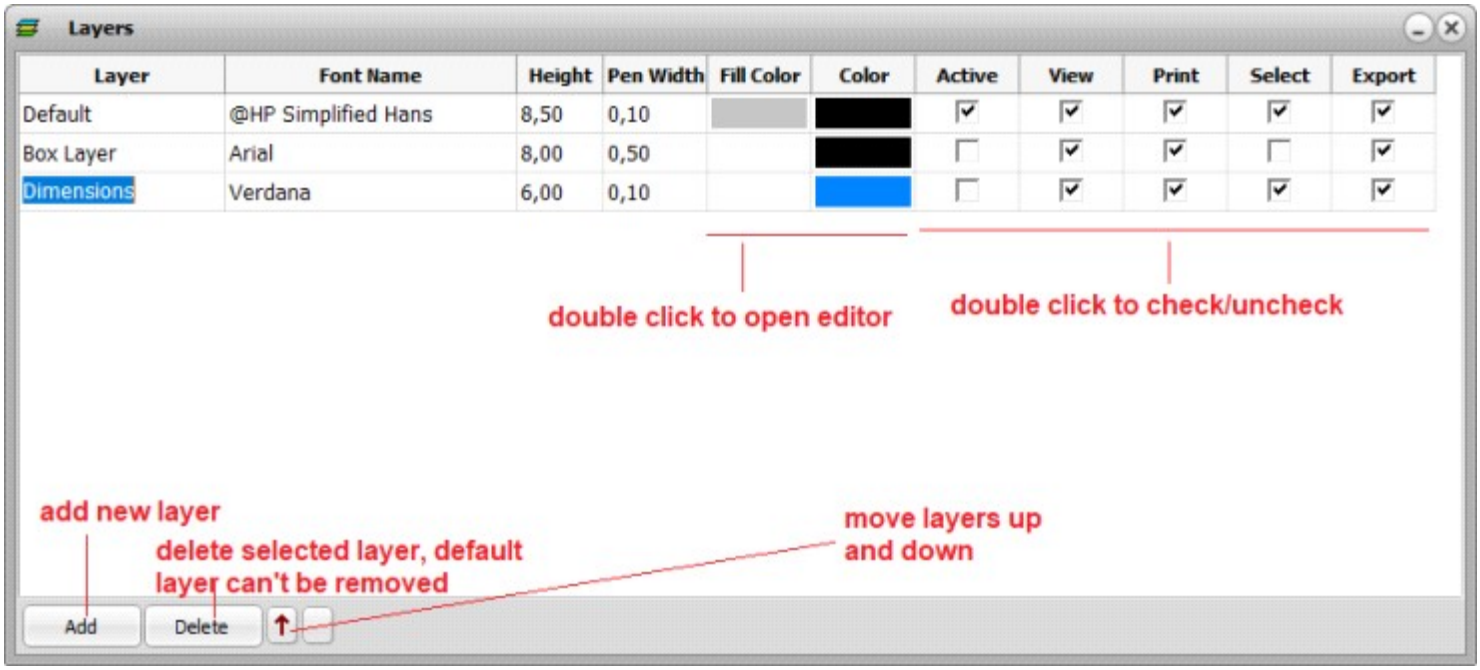


Fig.18 VCL Layer editor

If the option *loBackImageLayer* is selected, we can right-click on the layer in the layer editor and set an image as the background. If the option *loBackImageMap* is active, this can be a map obtained from an online service. Similarly, we will find a menu option to set the margins, in case we want the image to be a certain distance from the edges of the page.

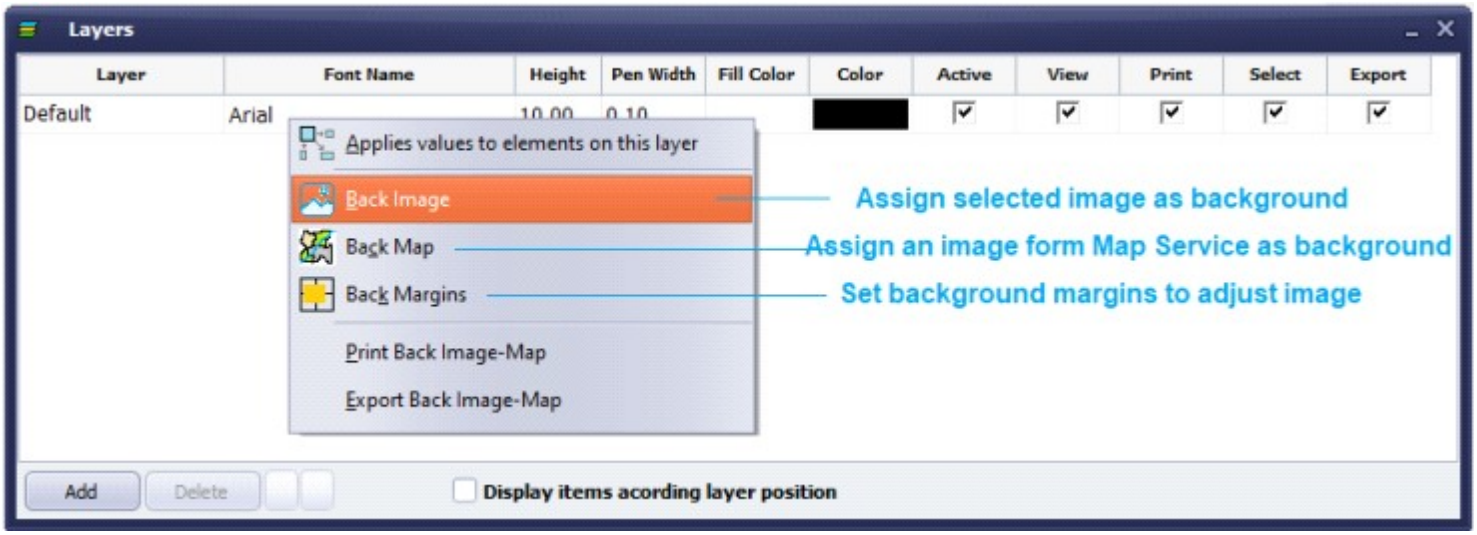


Fig.19 Select background image

EditProperties

We open the drawing properties editor, depending on the established mode we will see two types of editors, one for labels (*emLabelling*) and another for flowcharts (*emFlowChart*) and technical drawings (*emDrawing*).

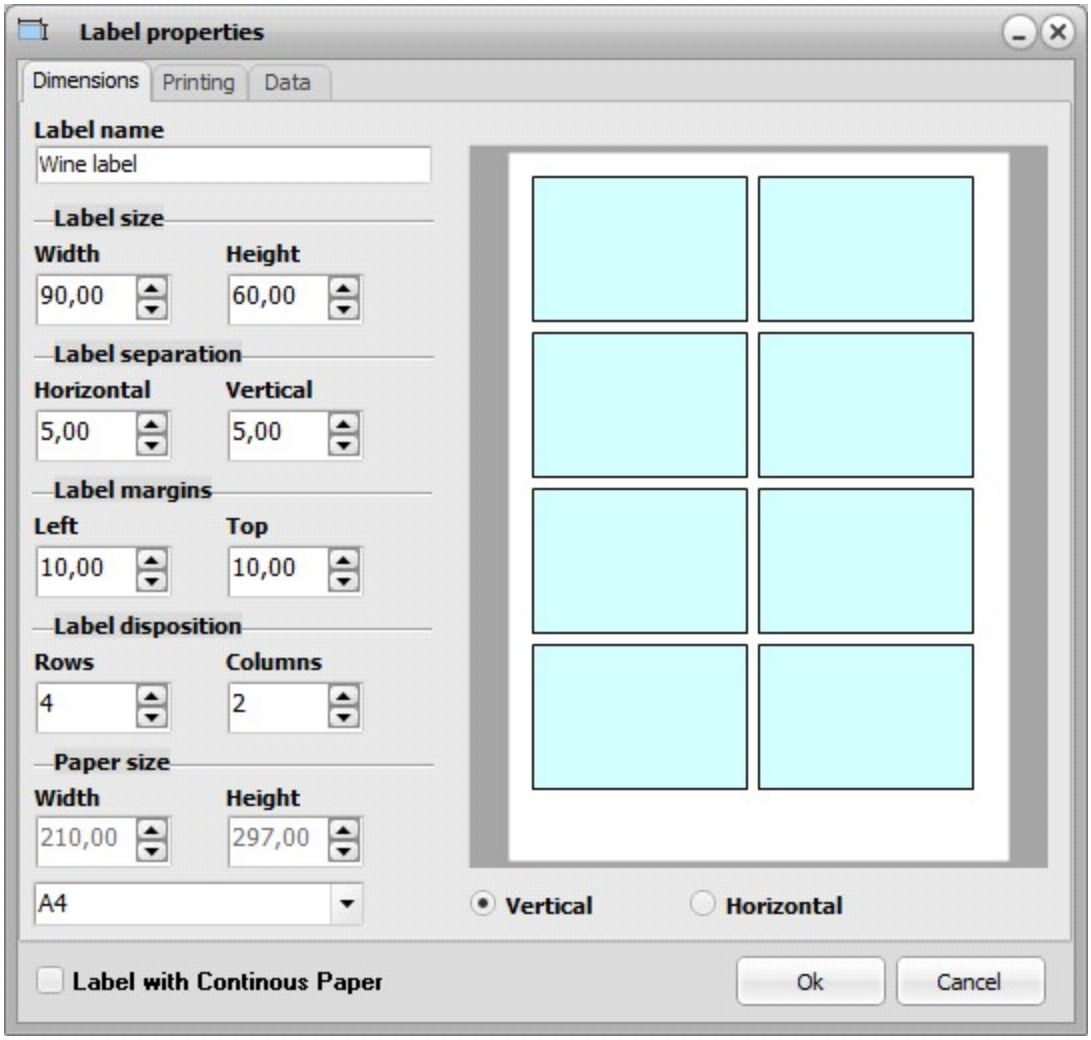


Fig.20 Editor properties for emLabelling mode

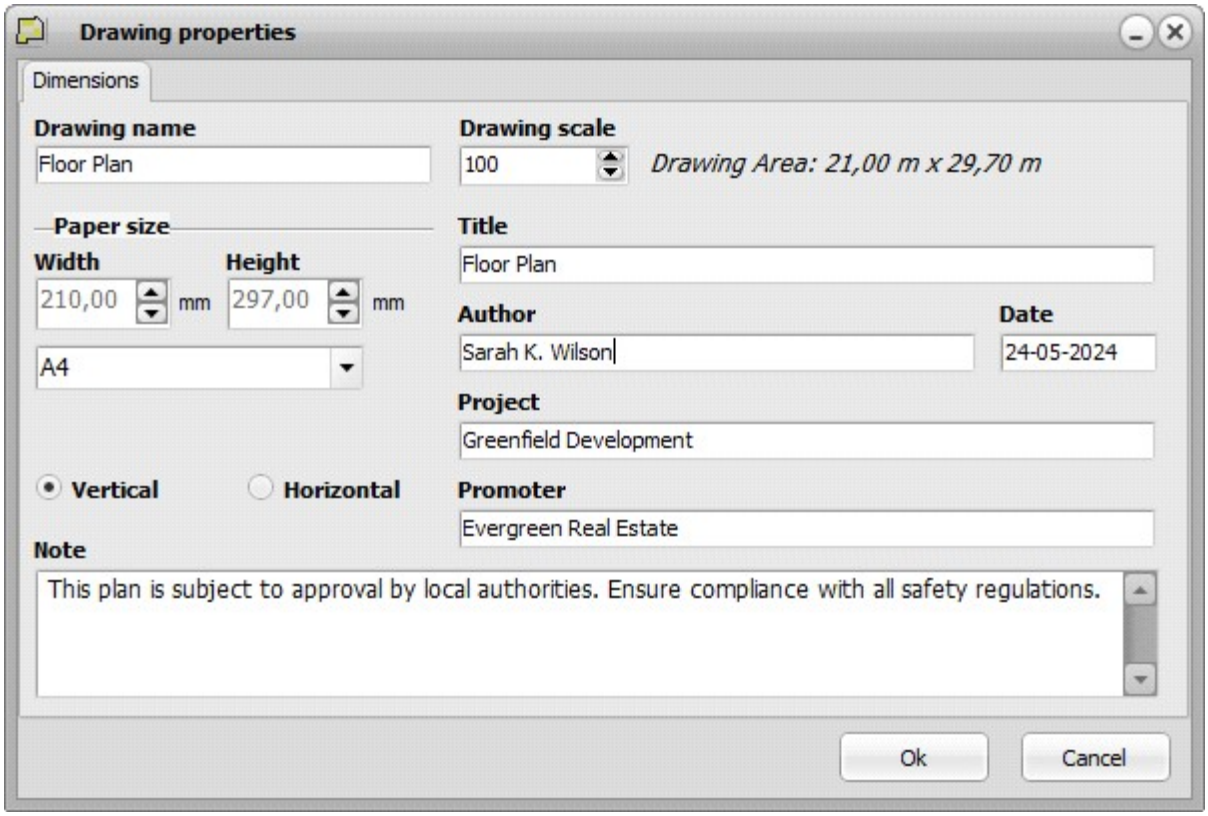


Fig. 21 Properties editor for drawing and flow chart mode

ExportDoc(aType: TMPLExport;const aFilename: TFileName;aPage: integer;aScale: single;aQuality: smallint;aCheckData,aTransparent: boolean;aFit: boolean = false)

We export asynchronous the current content in different formats: pdf, png, jpeg or gif. As parameters we can specify the drawing scale (if we want to increase or decrease the paper size, by default 1), in the case of png files we can indicate if the background is transparent, and for jpeg files the quality (top quality = 100). The aFit parameter is used to adjust the paper size to the size occupied by the elements.

FillBoxLayer(aReset: boolean = false)

In emFlowChart/emDrawing modes, it manages the dedicated title-block layer (lopBox): creates it if missing, optionally rebuilds frame primitives/texts when aReset = true, and otherwise updates dynamic fields (scale, title, author) from Measure.

```
MPLabel1.Measure.Title := 'Plant Layout - Rev B';
MPLabel1.Measure.Author := 'M. Taylor';
MPLabel1.Measure.DrawScale := 25;
MPLabel1.FillBoxLayer(False);
```

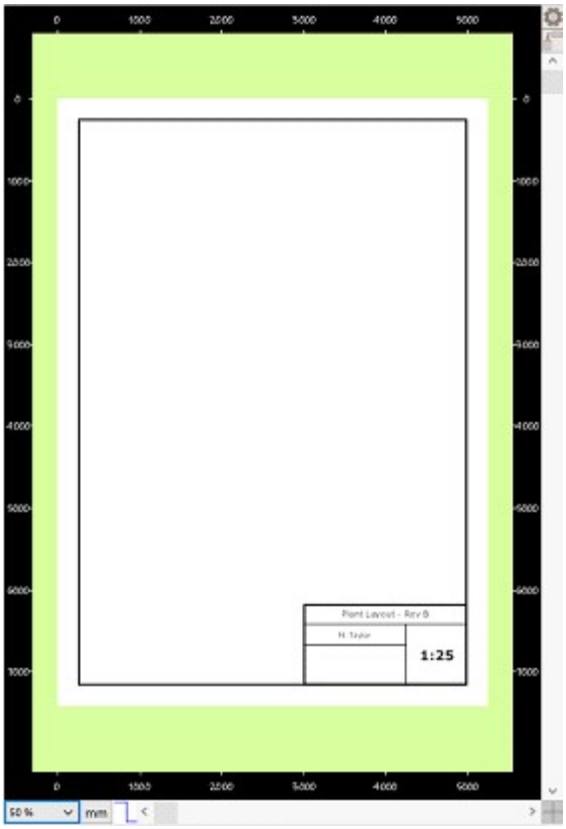


Fig.22 Title block

LoadFromFile(const aFilename: TFileName);

We load a previously saved file into the TMPLabel component, it can be in binary format (normally extension **.eti*) or a *json* .

```
procedure TFMain.butOpenClick(Sender: TObject);
begin
 opendialog.Filter := 'PLABEL Documents (*.eti)|*.eti';
 opendialog.DefaultExt := 'eti';
 if opendialog.Execute then MPLabel.LoadFromFile(opendialog.FileName);
end;
```

LoadFromStream(aStream: TStream)

We load label and drawing data into the editor from a stream, including symbols and all items, for example from a blob field in a database.

```
procedure TFMain.butLoadFromDBClick(Sender: TObject);
var
 st: TMemoryStream;
begin
 st:= TMemoryStream.Create;
  try
   CDLABELSFILE.SaveToStream(st);
   st.Position := 0;
   MPLabel.LoadFromStream(st);
```

```
  finally
    st.Free;
  end;
end;
```

PasteSelection

Pastes previously copied items from internal clipboard stream, recreating elements in paste mode.

Preview

We open the editor with a preview of the print. From this editor we can send to print, or save as pdf or image (png, jpeg and gif). When we save we can modify the size by applying a scale, as well as checking the box that we want to adjust the size to what the elements of the drawing occupy. In the case of png we can apply transparency by checking the box.

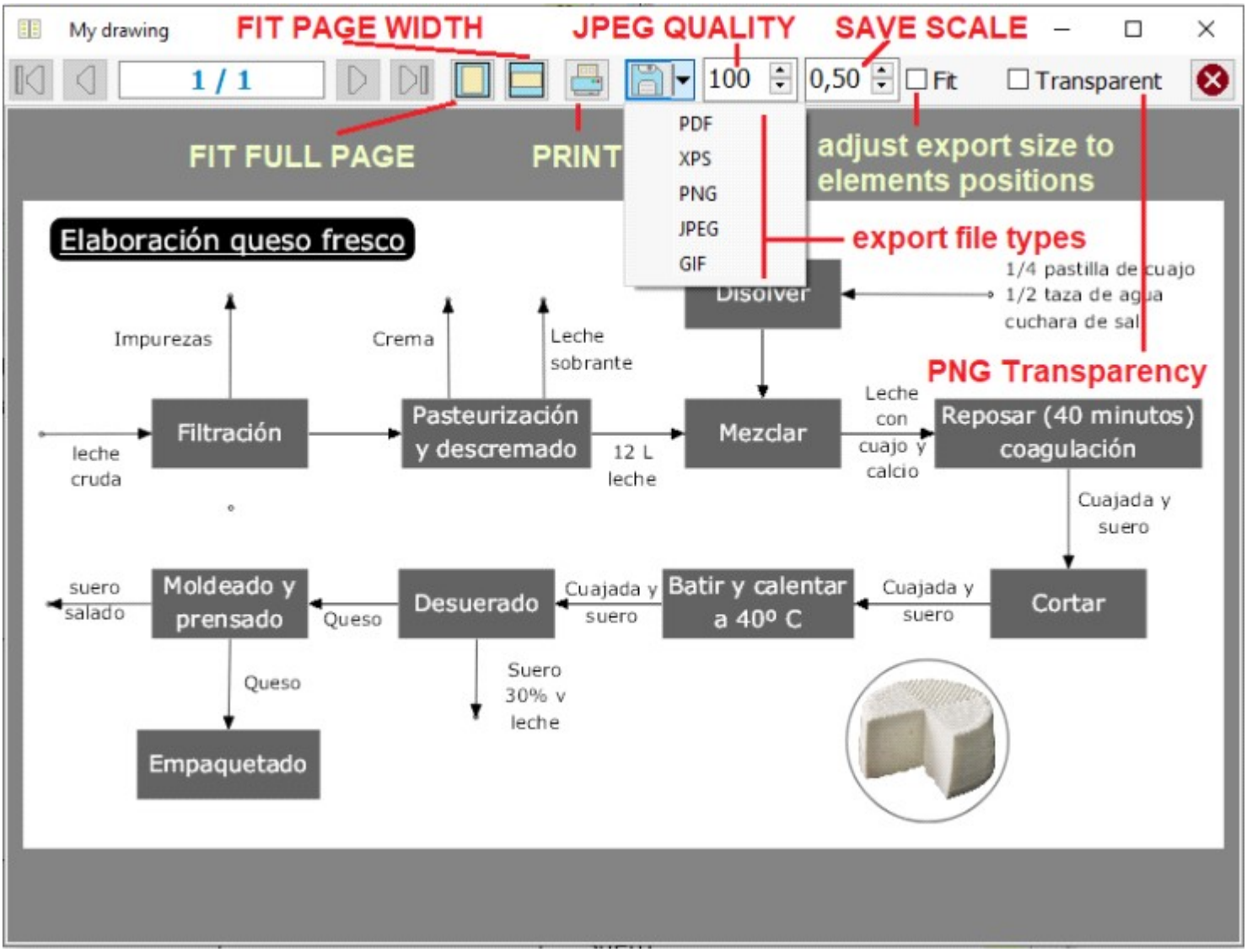


Fig.23 Preview

Print

We send the content of the editor to print taking into account whether or not the elements have the print

option checked, and whether the layer assigned to them has it checked. Validates print data and starts asynchronous printing using TPrintThread. We print with the printer indicated in *PrintInfo.PrinterIndex.*

```
procedure TFMain.butPrintClick(Sender: TObject);
begin
 MPLabel1.PrintInfo.PrinterIndex := cbPrinters.ItemIndex;
 MPLabel1.Print;
end;
```

SameWidth

When we have more than one element selected, it sets the width of the elements with the same width size as the first one.

SameHeight

When we have more than one element selected, it sets the height of the elements with the same height size as the first one.

SaveToFile(const aFilename: TFileName;aFormat: TMPLExt = lfBin)

We save the current content in a file, by default in binary format (the .eti extension is usually used), or as json (aFormat = lfJSON)

```
procedure TFMain.butSaveClick(Sender: TObject);
begin
 SaveDialog.Filter := 'PLABEL Documents (*.eti)|*.eti';
 SaveDialog.DefaultExt := 'eti';
 if SaveDialog.Execute then MPLabel.SaveToFile(SaveDialog.FileName);
end;
```

SaveSVG(const aFilename: TFileName;ExportType: TMPLExportSVG;Factor: single;attrID: boolean)

In version 3.0 the ExportType must be = svCurrentUnit, Factor = 1.

attrID = true id we want to indicate the id attribute in svg elements

```
procedure TForm3.butSaveSVGClick(Sender: TObject);
begin
 SaveDialog.Filter := 'SVG Files (*.svg)|*.svg';
 SaveDialog.DefaultExt := 'svg';
 if SaveDialog.Execute then MPLabel.SaveSVG(SaveDialog.FileName,svCurrentUnit,1,true);
end;
```

SaveToStream(aStream: TStream);

Writes complete label data (base data, properties, symbol library, items) in binary format to a stream, for example if we want to save in a blob field of a database.

```
procedure TFMain.butSaveToDBClick(Sender: TObject);
var
 st: TMemoryStream;
begin
 st:= TMemoryStream.Create;
  try
   MPLabel.SaveToStream(st);
   st.Position := 0;
   CDLABELS.Edit;
   CDLABELSFILE.LoadFromStream(st);
   CDLABELS.Post;
  finally
   st.Free;
  end;
end;
```

ElementByID

```
function ElementByID(const aID: string; aSelect: boolean = true): TMPLBaseItem;
```

We search for an element in the drawing by its ID, and with the parameter aSelect (default true) we indicate whether we want it to be selected.

6.7.3 Events

6.7.3.1 OnBeforePrint

```
procedure(Sender: TObject);
```

Before starting the printing process (printing thread).

6.7.3.2 OnAfterPrint

```
TNotifyEvent  procedure(Sender: TObject);
```

Upon successful completion of printing.

6.7.3.3 OnInfoMsg

```
TMPLInfoMsgEvent
procedure (Sender: TObject; const Action, State: string)
```

When the component notifies state/action changes for UI or logs. Action: current action of the editor. State: textual state of the editor.

6.7.3.4 OnReadSymEvent

```
procedure (Sender: TObject; const aFile: string; var aRead: boolean) of object
```

Symbol libraries are read from files found in the *LibraryFolder* folder of the **TMPLabelVCL or TMPLabelFMX** components. In this event we can filter the symbol files (extension .lyp) that we want to select. In this sample we exclude the symbols defined in the *restaurant.lyp* file:

```
procedure TFMain.MPLabelReadSymEvent (Sender: TObject; const aFile: string;
var aRead: Boolean);
begin
if (ExtractFileName (aFile) = 'restaurant .lyp') then aRead:= false;
end;
```

6.7.3.5 OnGetFontListEvent

```
TMPLGetFontListEvent = procedure (Sender: TObject; List: TStrings) of object;
```

With this event you can filter the text fonts to be displayed in the selection (List: list to be filled with font names), when editors (inspector, layers, table, etc.) need to load sources, for example in the **TMPLInspector** and the layer editor. If you need to do this you can implement the event like this:

```
procedure TFMain.MPLabelGetFontListEvent (Sender: TObject; List: TStrings);
var
 aList: TStringList;
begin
 aList:= TStringList.Create;
 try
  aList.Add('Arial');
  aList.Add('Tahoma');
  aList.Add('Verdana');
```

```
   List.Assign(aList);
  finally
   aList.Free;
   end;
 end;
```

6.7.3.6 OnUndoRedoEvent

```
TMPLUndoRedoEvent  procedure(Sender: TObject; emptyUndo, emptyRedo: boolean)
```

Following changes affecting Undo/Redo actions, for example to enable/disable buttons.

6.7.3.7 OnGetListValues

```
TMPLGetListValues  procedure(Sender: TObject; const aName: string; ListValues: TStrings; var aFill: b
```

aName: Name of the requested list. **ListValues**: List of values to be filled. **aFill**: Should be set to true if the list was loaded from the event.

To dynamically provide data list options

6.7.3.8 OnSelectItem

```
TMPLSelectItem  procedure(Sender: TObject; aItem: TMPLBaseItem)
```

aItem: selected item.

When an object enters selection.

6.7.3.9 OnUnSelectItem

```
TMPLSelectItem  procedure(Sender: TObject; aItem: TMPLBaseItem)
```

aItem: item that is removed from selection.

When an object is deselected.

OnLabelRotate

```
TNotifyEvent  procedure(Sender: TObject)
```

When label rotation (left/right) is performed, it is only possible in *emLabelling* editor mode.

6.8.TMPLInspector

The **TMPLInspector** component is used in conjunction with the **TMPLabelVCL** or **TMPLabelFMX** components (using Plabel property), and is used to edit the properties of the element or elements (common properties) that we have selected. It is an element with a fixed width of 240 pixels. In version 3.0 it has been designed from scratch to display the properties in a more compact way. If we have defined styles (VCL Styles) in the project, the component will use its colors.

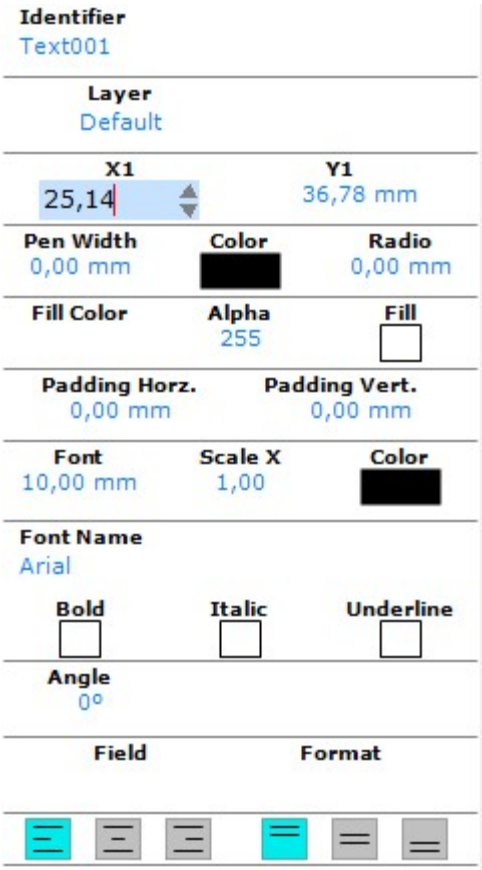


Fig.24 Inspector component

6.8.4 Properties

6.8.4.10 PLabel

Links the inspector to a **TMPLBaseLabel** (or descendant such as **TMPLabelVCL** / **TMPLabelFMX**) component. When a link is established the inspector subscribes as an *IMPObserver* and automatically refreshes whenever the selected element changes. Setting this property to nil detaches the inspector.

6.8.4.11 InspectPropBase

Controls which base property groups the inspector displays. Each flag enables or disables a category of properties shared by most element types. Available flags:

- iyID — element identifier
- iyLayer — layer assignment
- iyFont — font name, size and style
- iy1D — 1D barcode specific properties
- iy2D — 2D barcode properties
- iyBack — background fill
- iyTwoPoint — two-point geometry (start/end)
- iyPoint — single-point position
- iyBorder — border / outline
- iyAngle — rotation angle
- iyData — data-binding fields
- iyRadio — radius / diameter
- iyCenter — centre point
- iyAspect — aspect-ratio lock
- iyFrame — frame rectangle
- iyMemo — multiline text content
- iyPadding — internal padding
- iyAlignHorz — horizontal alignment
- iyAlignHorzVert — horizontal + vertical alignment
- iySize — width / height sizing
- iyWidthHeight — explicit width and height values
- iyBackBarcode — barcode background

6.8.4.12 InspectProp

Controls which element-specific property groups appear. Each flag corresponds to a particular element type. Available flags:

- iyLine — line element properties
- iyDataImage — data-bound image
- iyBarcode — barcode properties
- iyText — text / paragraph properties
- iyImage — static image properties
- iyEmdeded — embedded object properties
- iyLink — hyperlink properties
- iyTable — table properties
- iyPolyText — poly-text (rich multiline) properties
- iyRank — ranking element
- iyLevel — level indicator
- iyNutriScore — Nutri-Score element
- iyArrowLine — arrow / directed line
- iySymbol — symbol library element
- iySector — sector / arc element
- iyDim — dimension line properties
- iyDimStyle — dimension style properties
- iyPackage — package element properties

6.8.4.13 Options

Behavioural flags that modify how the inspector operates:

- ioEditTextOnCreate — automatically opens the text editor when a new text element is created, so the user can start typing immediately.
- ioMouseWheel — enables mouse-wheel scrolling inside the inspector panel.
- ioSelMapImage — when enabled, image selection uses a map-based picker instead of the standard file dialog. you must pick on it.

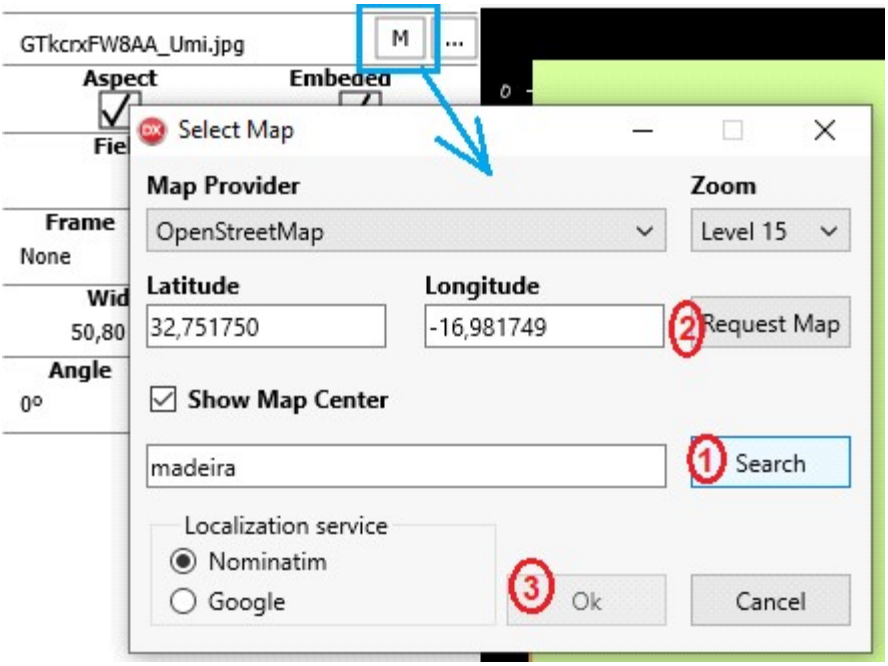


Fig. 25 Select map image button

6.8.4.14 BackColor

Background colour of the inspector panel. In VCL, when visual styles are active this value is overridden by the current theme's window colour.

- VCL: TColor — clWindow
- FMX: TAlphaColor — White

6.8.4.15 TitleColor

Colour used to draw property-name labels (the left column of each row).

- VCL: TColor — clBlack
- FMX: TAlphaColor — Black

6.8.4.16 ValueColor

Colour used to draw property values (the right column of each row).

- VCL: TColor — clNavy
- FMX: TAlphaColor — Navy

6.8.4.17 EditColor

Background colour of the inline edit field that appears when the user is editing a property value.

- VCL: TColor — clWhite
- FMX: TAlphaColor — White

6.8.4.18 SelColor

Highlight colour for the currently selected (focused) row in the inspector.

- VCL: TColor — clNavy
- FMX: TAlphaColor — Navy

6.8.4.19 SelButColor

Colour of the action button (e.g., the ellipsis "..." or drop-down arrow) when the row it belongs to is selected.

- VCL: TColor — clTeal
- FMX: TAlphaColor — Teal

6.8.4.20 ArrowColor

Colour of the small drop-down and navigation arrows drawn inside the inspector (e.g., the combo-box arrow indicator and scroll arrows).

- VCL: TColor — clGray
- FMX: TAlphaColor — Gray

6.9.TMPLEditor

TMPLEditor is the visual label/drawing/flowchart editor panel. It provides scrollbars, rulers, zoom, grid, snap and inline editing around a linked **TMPLabelVCL** / **TMPLabelFMX** component. It exists in two framework-specific versions — VCL (VCL\MPL.VCL.Editor.pas) and FMX (FMX\MPL.FMX.Editor.pas) — both deriving from TMPLBaseEditor.

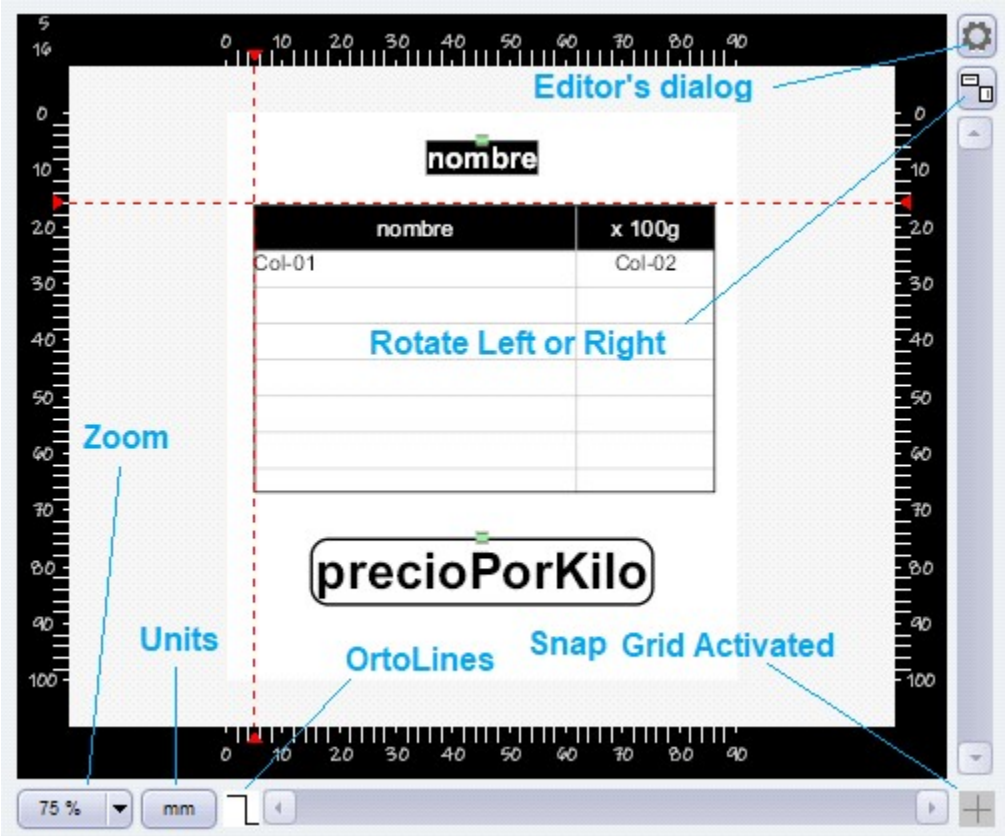


Fig.26 Editor

6.9.5 Properties

Plabel

Links the editor to a label component. When linked the editor subscribes as an IMPObserver and displays the label's content with scrollbars, rulers and editing capabilities. Setting to nil detaches the editor.

- VCL: TMPLabelVCL
- FMX: TMPLabelFMX

Zoom

Current zoom level. Predefined values: zm25, zm50, zm75, zm100, zm125, zm150, zm175, zm200, zm250, zm300, zm350, zm400, zm450, zm500, zmWidth (fit width), zmHeight (fit height), zmFitAll (fit entire page). Within the editor itself, and if we enable the option, we can display a selector to choose the zoom level.

Rulers

Configures the rulers displayed along the editor edges and its functionality.

Property	Type / Default	Descripction
Visible	TMPLShowRulers(set of TMPLRulerVisible)Default: [rvLeft, rvTop]	Which rulers to display. Values: rvLeft, rvTop, rvRight, rvBottom.
Font	TFont	Font used for ruler tick labels.
FontColor	FMX only: TAlphaColor — Black	Colour of the ruler tick labels. In VCL the font colour is set via the Font property itself. FMX
Color	VCL: TColor — clSilver. FMX: TAlphaColor — Silver	Background colour of the ruler bar.
ToColor	VCL: TColor — clWhite. FMX: TAlphaColor — White	Gradient end colour (for gradient-style rulers).
TickColor	VCL: TColor — clBlack. FMX: TAlphaColor — Black	Colour of the tick marks on the ruler.
MarkColor	VCL: TColor — clRed. FMX: TAlphaColor — Red	Colour of the current-position marker line on the ruler.
SizeColor	VCL: TColor — clNavy. FMX: TAlphaColor — Navy	Colour of the selection-size indicator on the ruler.
Size	Word	Width (or height) of the ruler bar in pixels.
ShowSize	Boolean Default: False	When True, shows the width/height of the current selection on the ruler.
ShowAlign	Boolean Default: False	When True, enables draggable alignment guides from the rulers.
Style	TMPLRulerStyle Default: rsStyle01	Visual style of the ruler: rsStyle01 (flat) or rsStyle02 (gradient).

Table21 Rulers properties

Grid

Configures the background grid.

Property	Type / Default	Description
Style	TMPLGridDraw Default: gdNone	Grid drawing style: gdNone (hidden), gdPoints (dot grid) or gdLines (line grid).
IncX	Single	Horizontal grid spacing in the current unit system.
IncY	Single	Vertical grid spacing in the current unit system.
StepColor	VCL: TColor — clSilver. FMX: TAlphaColor — Silver	Colour of the minor grid steps.
UnitsColor	VCL: TColor — clGray. FMX: TAlphaColor — Gray	Colour of the major (unit-boundary) grid lines.

Table22 Editor's Grid property

SelectionColors

Customises the colours used for selection handles, drawing feedback and reference lines.

Property	Default (VCL / FMX)	Description
Center	Blue	Colour of the centre handle of a selected element.
Radio	Lime	Colour of the radius/rotation handle.
SizeColor	Red	Colour of the sizing handles.
FixColor	Gray	Colour of fixed (locked) handles.
SelectionRect	Blue	Colour of the rubber-band selection rectangle.
SelectionItem	Silver	Overlay colour on selected items.
Draw	Teal	Colour of the active drawing line/shape while being drawn.
Point	Red	Colour of control points during drawing.
Bezier	Crimson	Colour of Bézier curve control handles.
Link	Orange	Colour of link/connector lines.
Align	Azure	Colour of alignment guide lines.
AlignText	Black	Colour of alignment distance labels.
LinkPoint	VCL: LemonChiffon. FMX: Cornsilk	Colour of link-point indicators.
RefLine	Lavender	Colour of reference alignment lines.
RefPoint	Crimson	Colour of reference points on alignment lines.

Table23 SelectionColors property options

MoveX

Horizontal scroll offset of the label inside the editor, expressed in the current unit system, when move objects with keyboard arrows. Read/write at run time; persisted at design time.

MoveY

Vertical scroll offset of the label inside the editor, expressed in the current unit system, when move objects with keyboard arrows.

Snap

Snap modes active during drawing and dragging:

snPoint	snap to existing points elements (TMPLPoint)
snGrid	snap to grid intersections
snMiddle	snap to midpoints of lines, arrows and segments (polylines and polygons)
snCenter	snap to element centres (circle, ellipse, rectangle)
snEnd	snap to endpoints (lines, arrow, polyline and polygon vertices)
snRefSymbol	snap to symbol reference points
snCross	snap to intersection (crossing) points of objetcs between them
snTangent	snap tangent to arcs and circles

snPerpendicular snap perpendicular to segments

Table24 Snap options

OrtoLines

When True, constrains drawing to orthogonal (horizontal / vertical) lines. You can activate this option by pressing the F8 key

Margin

Margin in screen pixels between the label boundary and the editor panel edges. Provides visual breathing room around the label.

MarginColor

Fill colour of the margin area surrounding the label.

Gap

Minimum distance (screen pixels) used to detect proximity to control/reference points when attaching drawing actions. A larger value makes it easier to grab handles.

SizeRef

Size in screen pixels of the reference-line handles drawn on selected elements.

Label position

With this property you can adjust the position of the label in the editor, aligning it in the top left corner (lbTopLeft), or centering it in the middle (lbCenter) .

Fig.27 Align label topleft

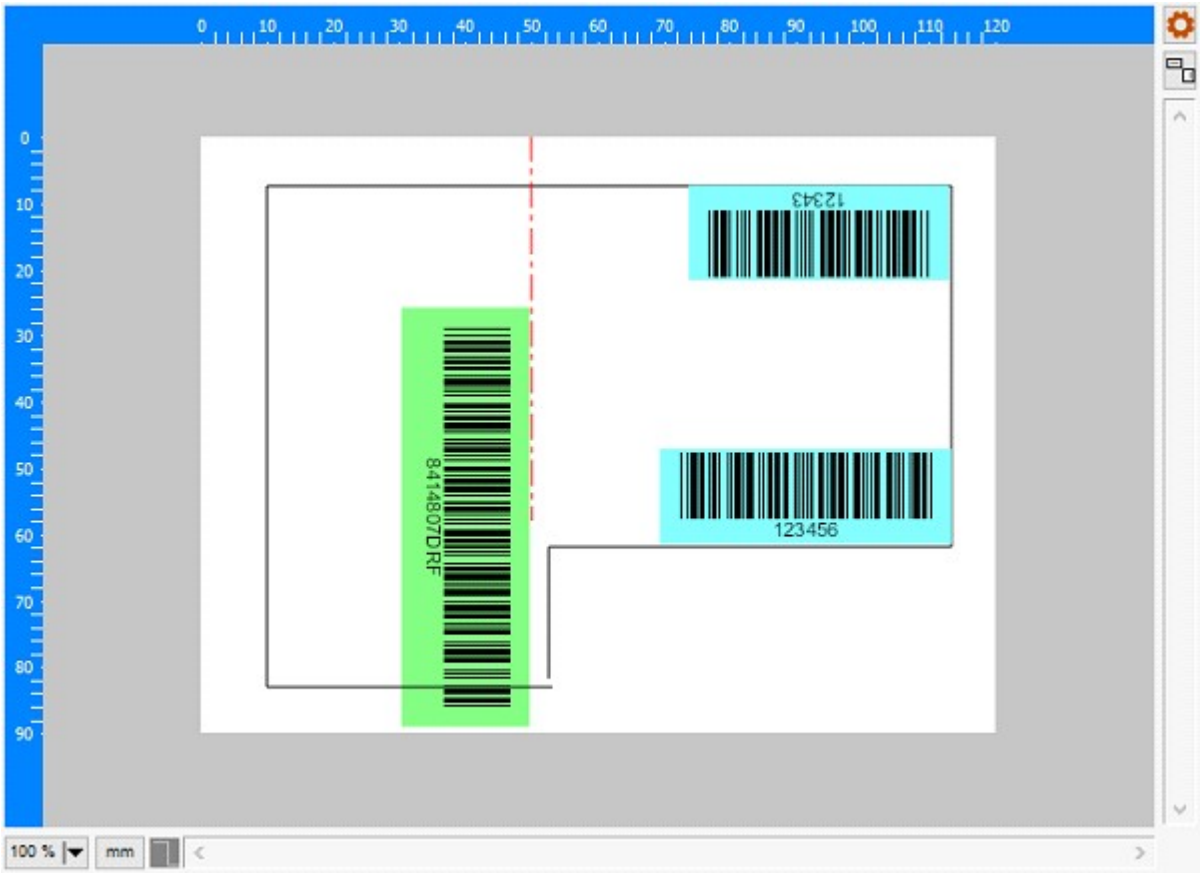


Fig.28 Align label centered

Options

eoZoom	Show zoom selection
eoUnits	Show units button
eoSettings	Show button to open editor settings
eoRotate	Show button to rotate the label left or right (only available in Labelling mode)
eoShowOrto	Show indicator that we have Orthogonal mode enabled for drawing lines
eoShowSnapToGrid	Show indicator that we have enabled snap to grid
eoInput	You can enter coordinates (absolute 10,20 or relative @5,10), of the current draw tool point, and the text box appears at the top of the editor (press enter to validate)
eoMouseWheel	When enabled, it controls: - Vertical scroll (normal wheel) - Horizontal scroll (Shift + wheel) - Zoom (Ctrl + wheel) If removed from the options, the mouse wheel is completely ignored.
eoEditOnDblClick	This option controls whether double-clicking certain elements (Paragraph, PolyText, etc.) opens the editor for some of their properties
eoDrawLinkPoints	When active, when drawing symbols that have defined links, visual markers (small dots/circles) are rendered in the positions where links can be connected to the symbol.
eoRichToolBar	Displays a button bar above the TMPLRichText element to facilitate inline editing.
eoResizeDataText	It displays an additional resizing handle on text linked to selected data. An extra point of type is added, allowing the user to drag to change the text's display width, because sometimes it's larger than necessary, such as when displaying the field name.

Table25 Editor options

EnabledMouseFilter

When True, enables a mouse-coordinate filter. Added in version 3.0.1.

ShowRefLines

When True, reference alignment lines are drawn while moving or resizing elements. Added in version 3.0.1.

6.9.6 Methods

BeginUpdate

Suspends visual updates to the editor. Call before making a batch of changes to avoid flickering.

EndUpdate

Resumes visual updates after a BeginUpdate call and triggers a repaint.

EditSettings

Opens the dialog box to edit the editor's properties. Alternatively, a button can be added to the active editor to open it.

Fig.29 Editor's settings dialog

AlignSelectedLeft

Aligns all currently selected elements to the leftmost edge of the selection (Show align lines option activated).

AlignSelectedRight

Aligns all currently selected elements to the rightmost edge of the selection (Show align lines option activated).

AlignSelectedTop

Aligns all currently selected elements to the topmost edge of the selection (Show align lines option

activated).

AlignSelectedBottom

Aligns all currently selected elements to the bottommost edge of the selection (Show align lines option activated).

ScrollToItem

```
procedure TMPLBaseEditor.ScrollToItem(aItem: TMPLBaseItem);
```

If the element we pass as a parameter is not visible in the editor, this method will center it on the screen. In this example we move the editor view to see the element with ID 'test'

```
procedure TForm3.butViewItemClick(Sender: TObject);
begin
 MPLEditor1.ScrollToItem(MPLabel1.ElementByID('test'));
end;
```

6.9.7 Events

6.9.7.21 OnMouseDown

```
TMPLMouseEvent = procedure(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Single)
```

Fires when the user presses a mouse button inside the editor surface. Coordinates X, Y are in the label's coordinate system (current units).

6.9.7.22 OnMouseUp

```
TMPLMouseEvent = procedure(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Single)
```

Fires when the user releases a mouse button inside the editor surface.

6.9.7.23 OnMouseMove

```
TMPLMouseMoveEvent = procedure(Sender: TObject; Shift: TShiftState; X, Y: Single)
```

Fires as the mouse moves over the editor surface. Useful for displaying coordinates or custom cursor feedback.

7.Create Symbols

One of the elements that we can insert into the drawings is a predefined symbol, symbols are a series of drawing instructions (line, circle, text, etc.) that we execute in the order in which they have been created, to which we can insert actions to change the color of the lines or backgrounds, or the type of text. When we create a symbol we indicate the type of measurement in which it is defined, this is important to take into account since it will be affected by the scale and units of the drawing. For example, if we create a symbol in meters of a square that is 1 x 1 m, it will be outside the margins of the drawing if we are using millimeters and a 1:1 scale on an A4 paper, however, if we work in meters and a 1:10 scale, we will be able to see the square in the drawing.

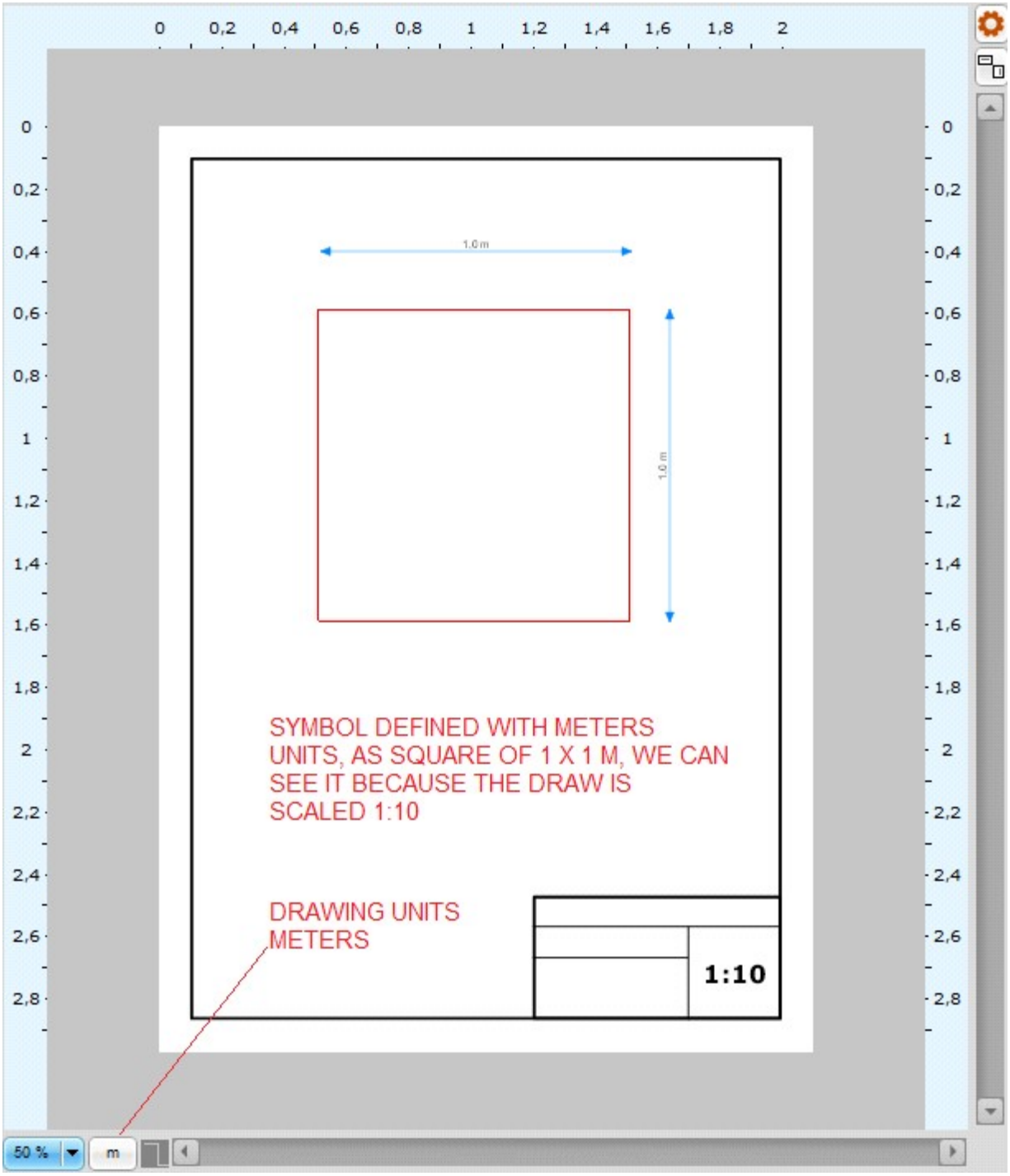


Fig.30 Symbol defined as square with 1 x 1 m

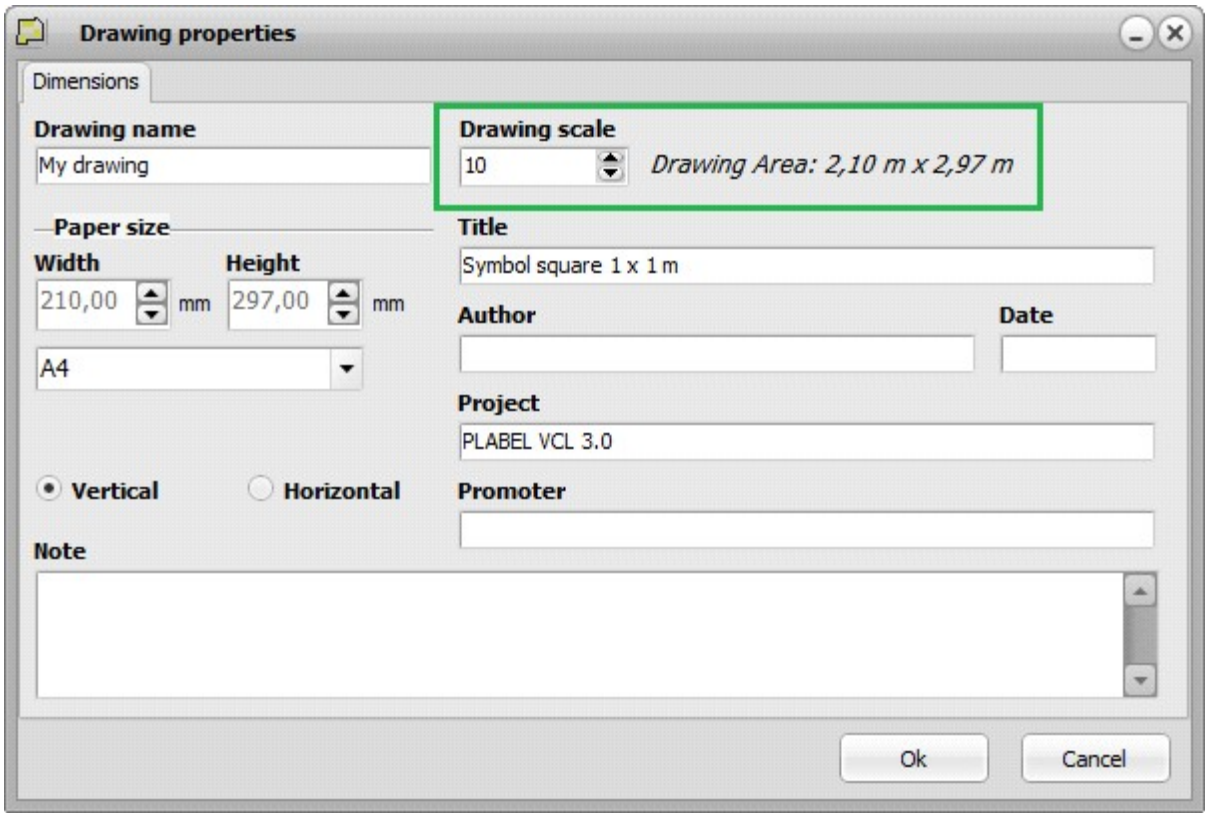


Fig.31 Drawing properties, setting 1:10 scale

Symbols can be created by code, and they can be assigned a name and grouped into categories. Afterwards, the collection of symbols can be saved in a file. When we click on insert symbol on the drawing, a dialog box opens where we can see the symbols in the symbols folder.

Let's imagine that we want to define a north arrow symbol to locate on maps.

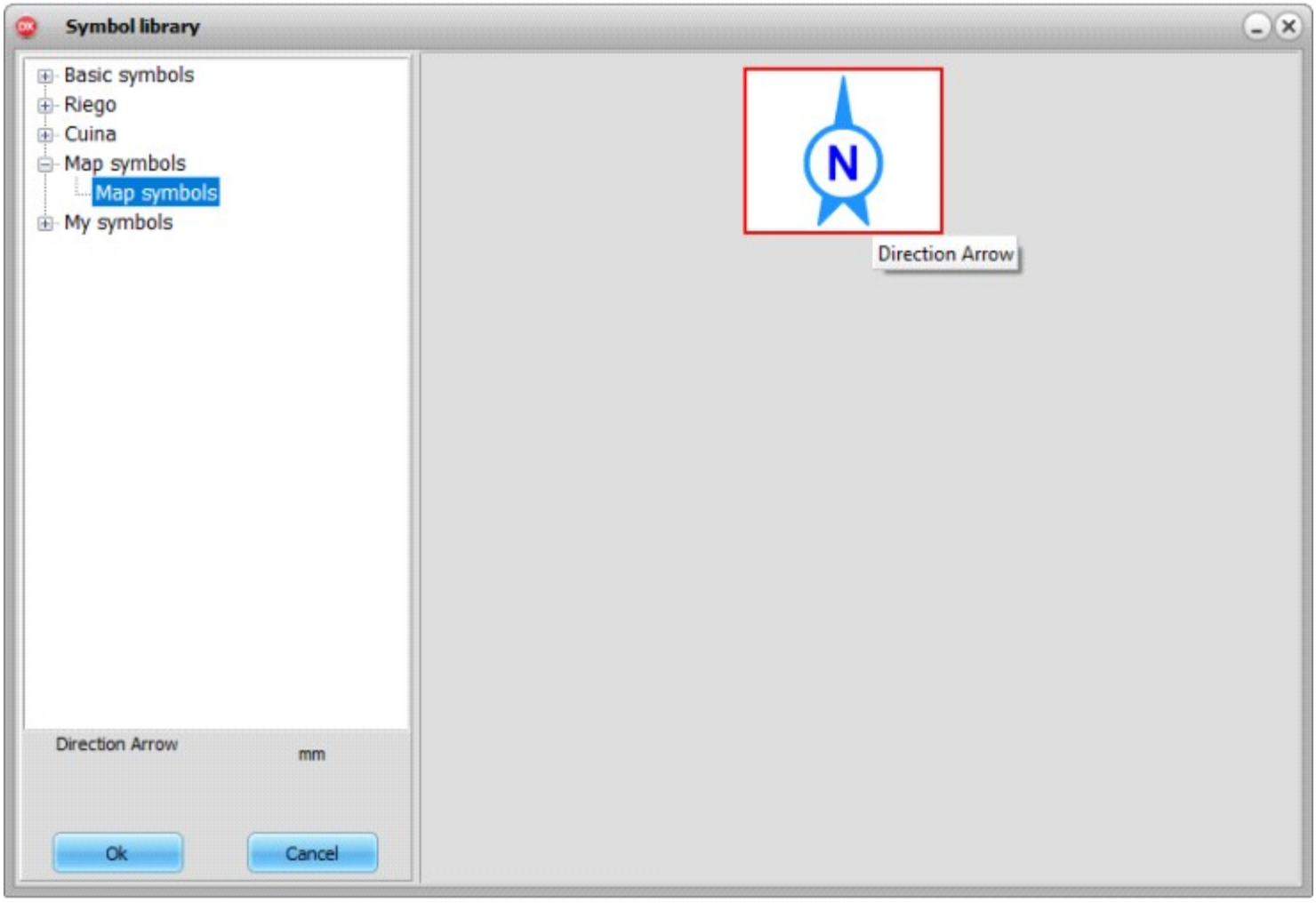


Fig.32 Map direction arrow

In this example we create a library object, where we can add one or more symbols, finally we persist the symbol library in a file. We can create this Direction Arrow symbol by code in this way:

```
procedure TForm3.ButCreateSymbolClick(Sender: TObject);
var
 lib: TMPLLibrary;
begin
 lib:= TMPLLibrary.Create;
 lib.Category := 'Map symbols';
 lib.AddSymbol('Direction Arrow','Map symbols','',unMM,0,0)
       .SetNotDrawScale(true)
       .Filled(true)
       .LineType(iiSolid)
       .LineWidth(0.1)
       .ForeColor(TAlphaColorRec.Dodgerblue)
       .BackColor(TAlphaColorRec.Dodgerblue)
       .Polygon([0,-7,2,5,0,3,-2,5,0,-7])
       .LineWidth(0.5)
       .BackColor(TAlphaColorRec.White)
       .Circle(0,0,3)
       .FontColor(TAlphaColorRec.Blue)
       .FontName('Arial')
       .FontHeight(4)
       .FontBold(true)
       .Text(0,0,'N',TMPLHorzAlign.haCenter,TMPLVertAlign.vaCenter);
 try
  if SaveDialog1.Execute then lib.SaveToFile(SaveDialog1.FileName);
 finally
  lib.Free;
 end;
end;
```

When we create a symbol we indicate the units and the insertion point, which serves as a reference when we express the coordinates of the various objects. If we define them with millimeters and with the NotDrawScale property to true, the symbol will be drawn according to the paper units, without taking into account the drawing units or the scale. When we insert a symbol on the drawing, we can apply a scale and a rotation angle to it. When we open the symbol selector, all the libraries (files with the lyp extension) that we have saved in the folder defined in the LibraryFolder property of the TMPLabel component are read.

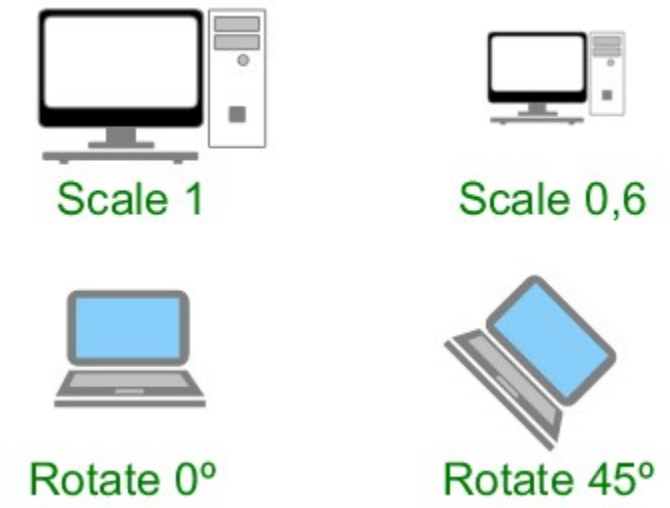


Fig.33 Scale and rotate symbol

8.List of figures

9.List of tables