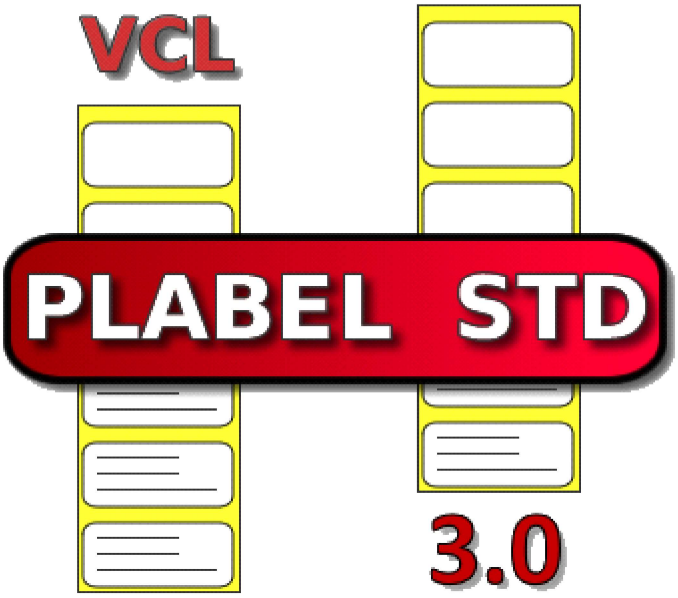
PLABEL VCL STD



Developer Document

1.0.1

September 2024

1.Page index

2.Introduction

The PLABEL VCL STD package is a series of components to add label editing and printing to your Delphi VCL programs, it's available for 32 and 64 bit versions. This version 3.0, is based on previous versions, but has been rewritten from scratch, is available for the Seattle, Berlin, Tokyo, Sydney, Rio and Alexandria versions. Now the editor has three modes, apart from editing labels we can design flowcharts and make CAD-type technical drawings.

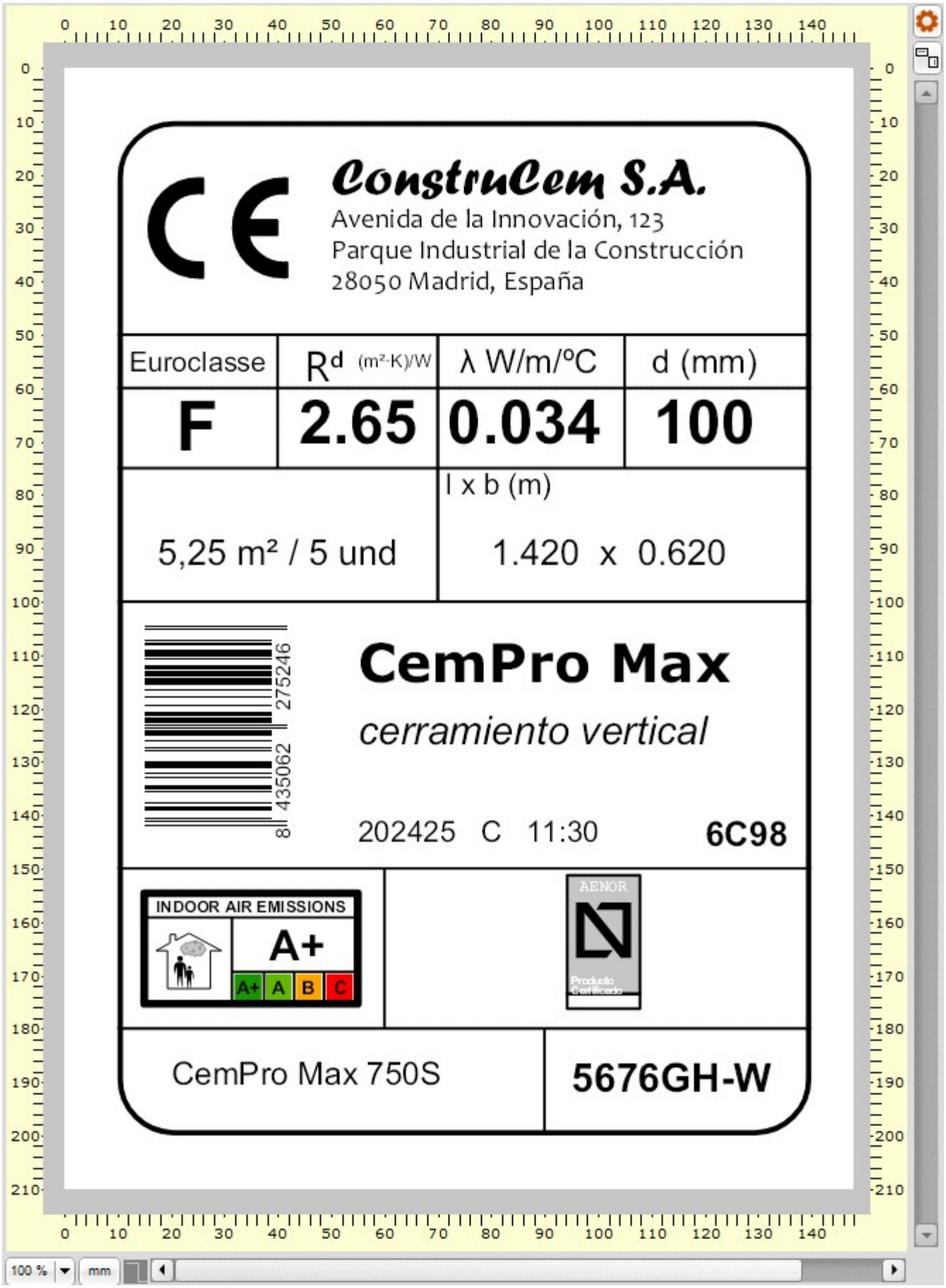


Fig.1 Label editor

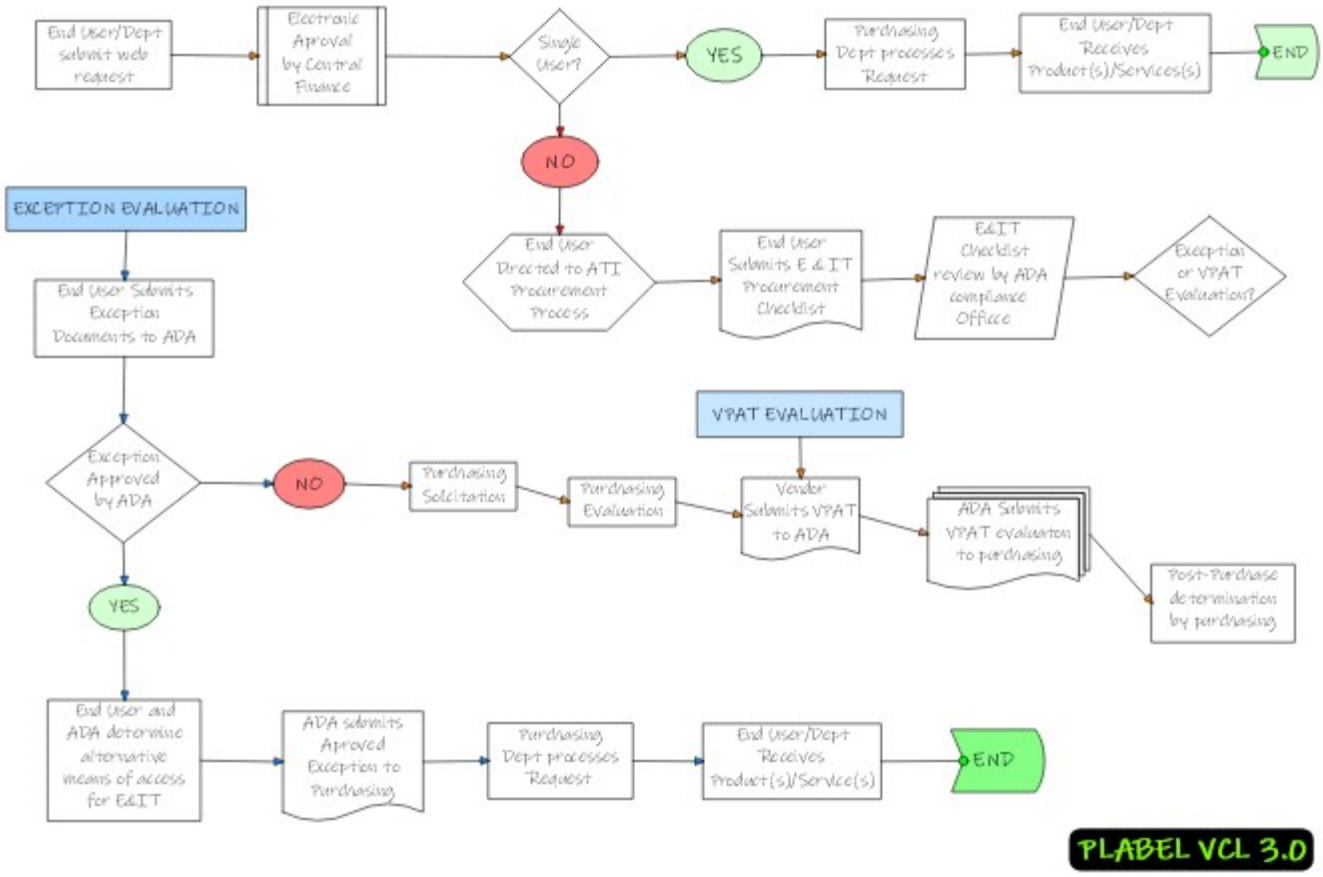


Fig.2 Flow chart

Fig.3 Technical drawing example

3.Installation

3.1 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 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 STD components can be done in two ways:

3.2 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.

3.3 Manual installation

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).

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**, **TMPLEditor** and **TMPLSpinEditFloat** components, and finally when opening the third project run Build and Install which will install the **TMPLabel** component in the PLABEL VCL STD tab.

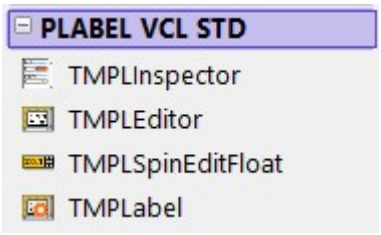


Fig.4 PLABEL VCL components

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 TMPLabel base component
TMPLInspector	Visual component to view and modify the properties of selected elements in the editor, connects to the TMPLabel base component
TMPLabel	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 and print files.

Table1 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_0\Source;
- C:\Users\Public\Documents\Mesurasoft\Plabel\VCL\3_0\Source\VCL;
- C:\Users\Public\Documents\Mesurasoft\Plabel\VCL\3_0\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.

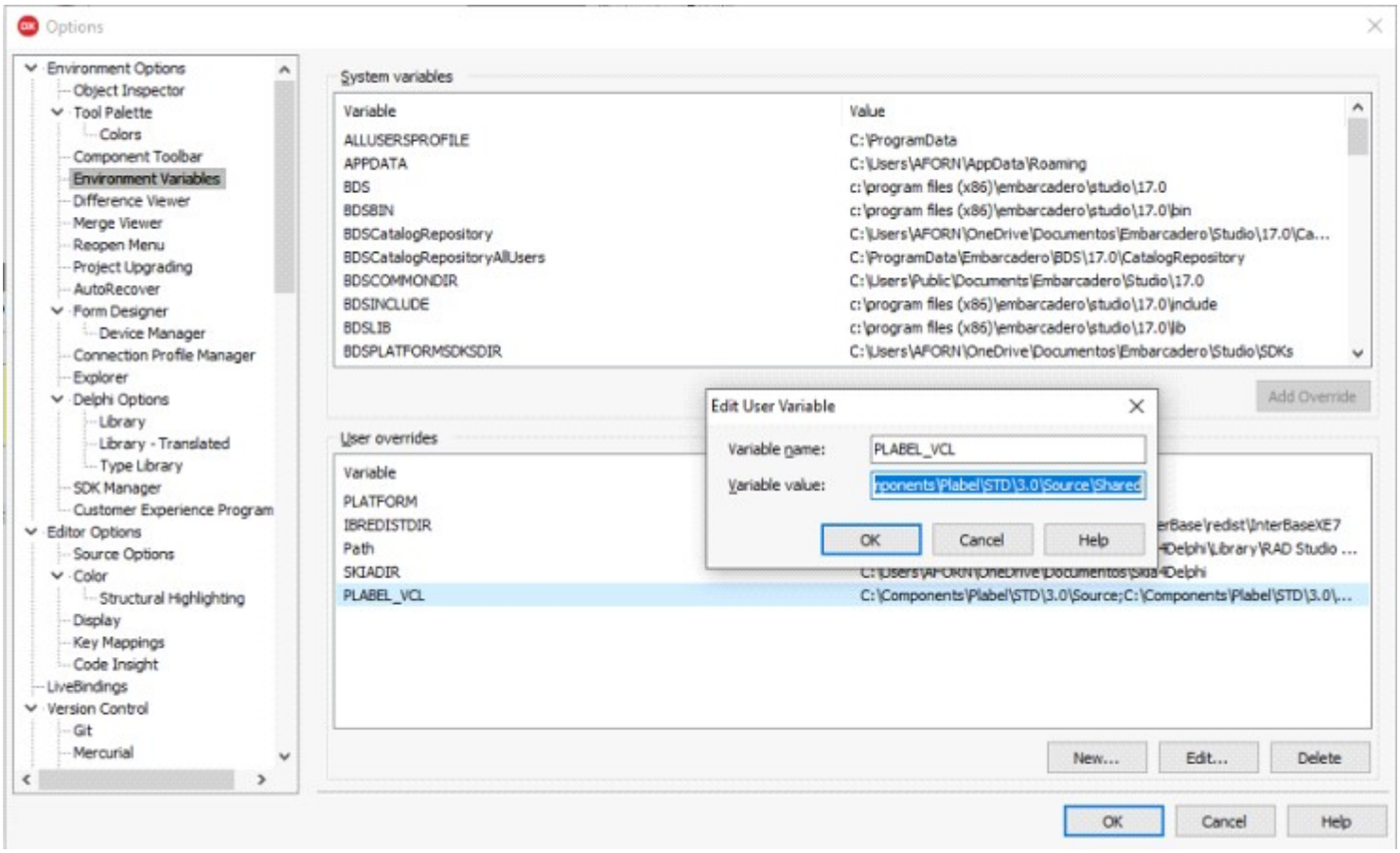


Fig.5 Define new variable

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

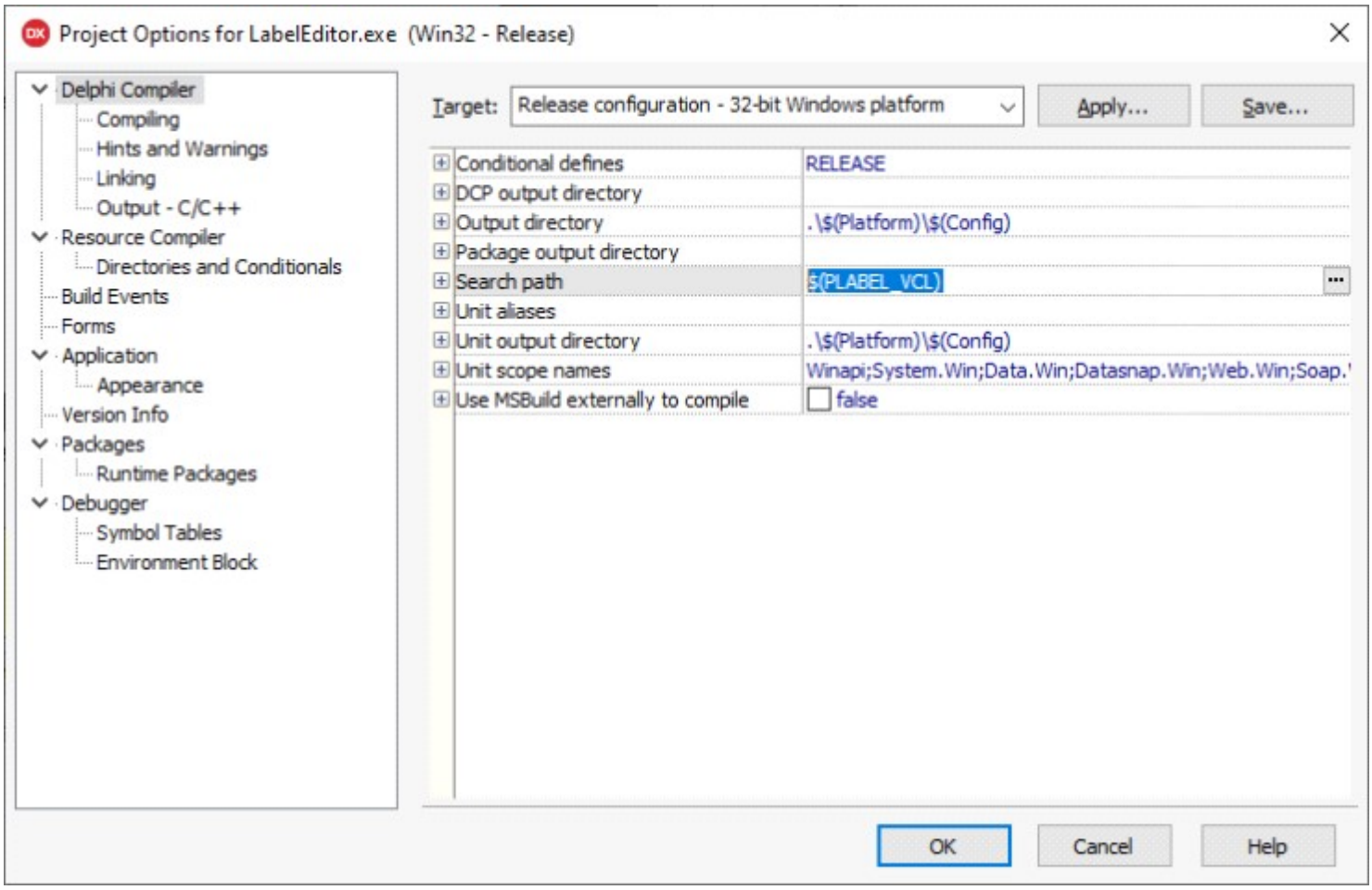


Fig.6 Search path variable

4.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 a ini file that we will assign to the LanguageFileName property of the **TMPLabel** component. If we do not specify a file, the default language will be English.

5.Components

5.4.TMPLabel

5.4.1 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, eaBarcode, eaPDF417,eaDataMatrix,eaQRCode,eaImage,eaFlow,eaLink,eaPolyLink,eaTable,eaPolyText,eaRank, eaNutriScore,eaLevel,eaSymbol,eaDim,eaSectorPR, eaSector2P, eaSector3P, eaPackage.

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

BackSurf.Color

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

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.

5.4.2 Methods

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.

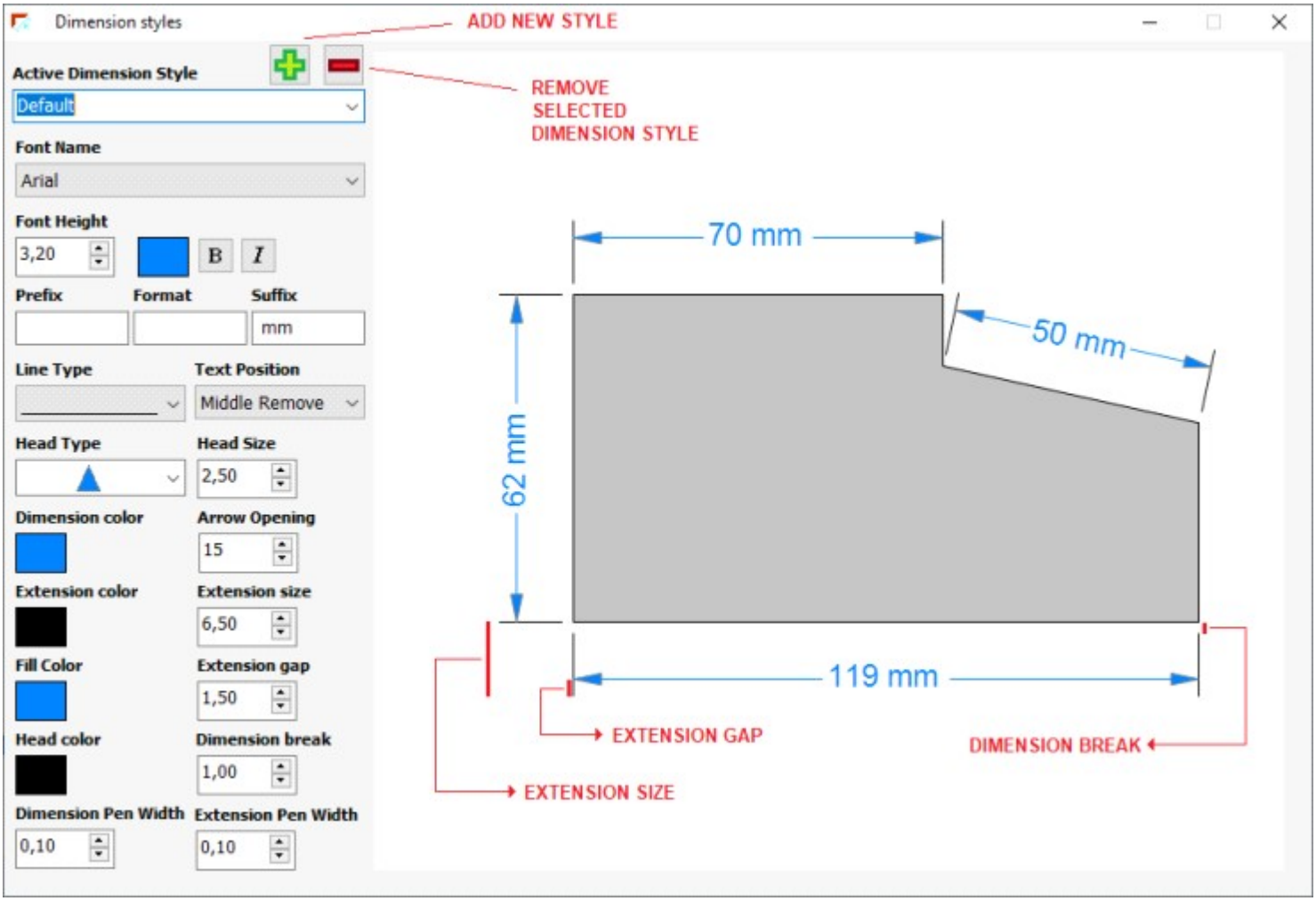


Fig.7 Dimension styles editor

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.

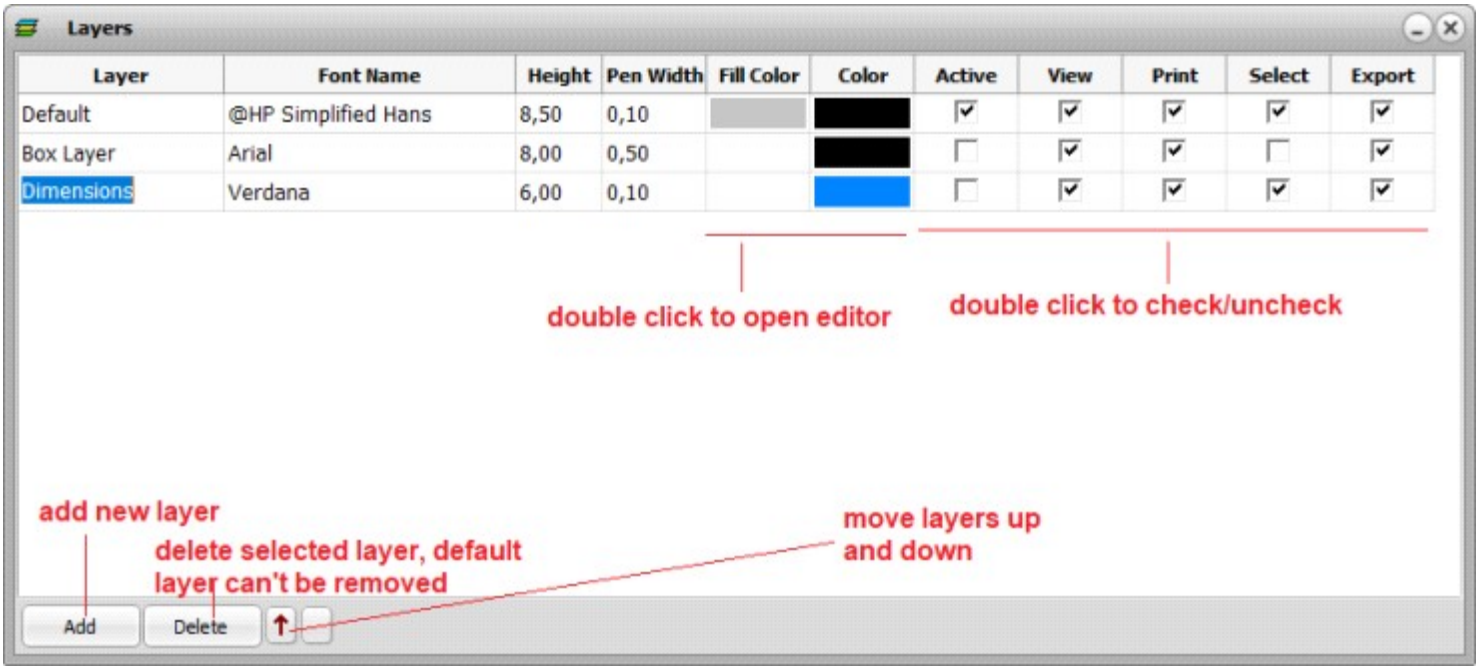


Fig.8 Layer editor

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*).

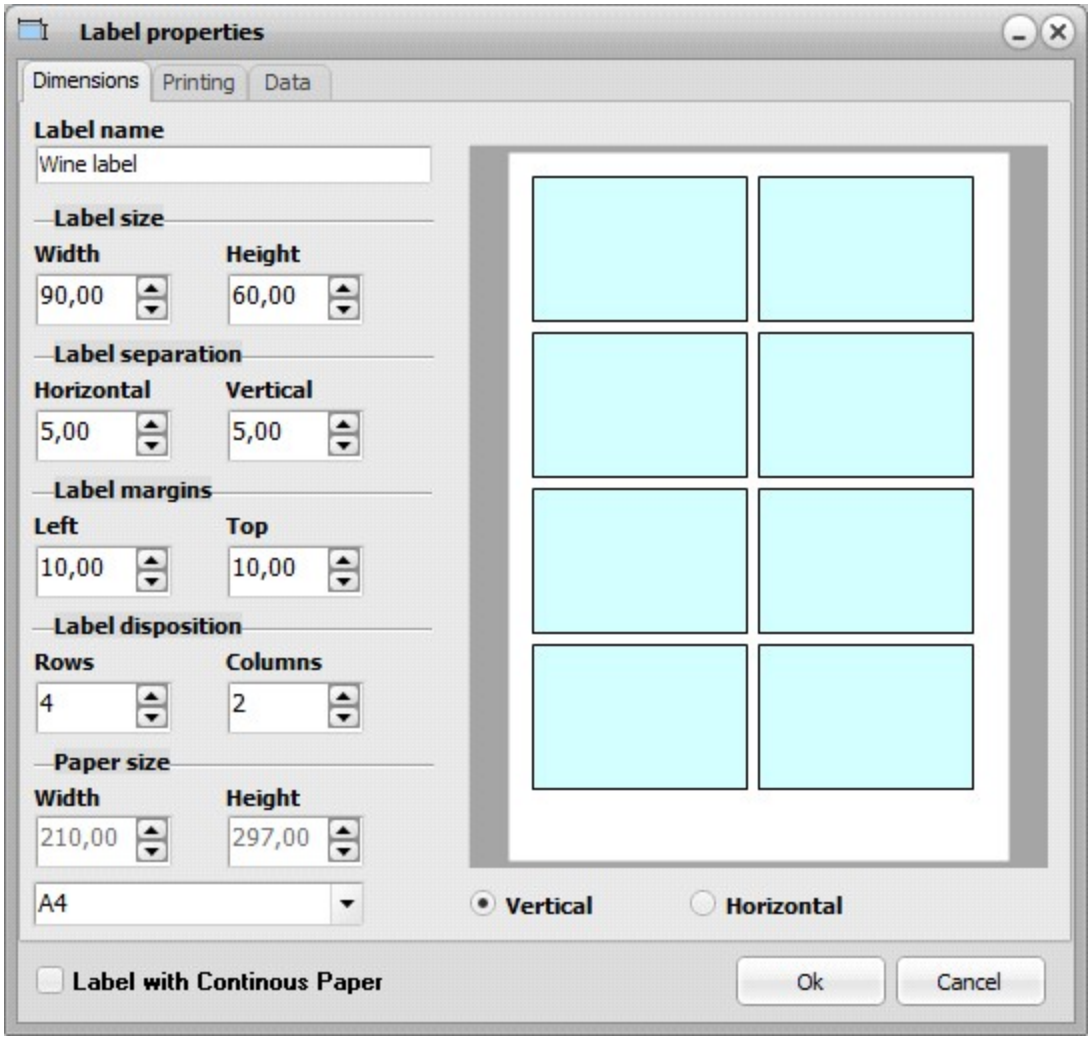


Fig.9 Editor properties for emLabelling mode

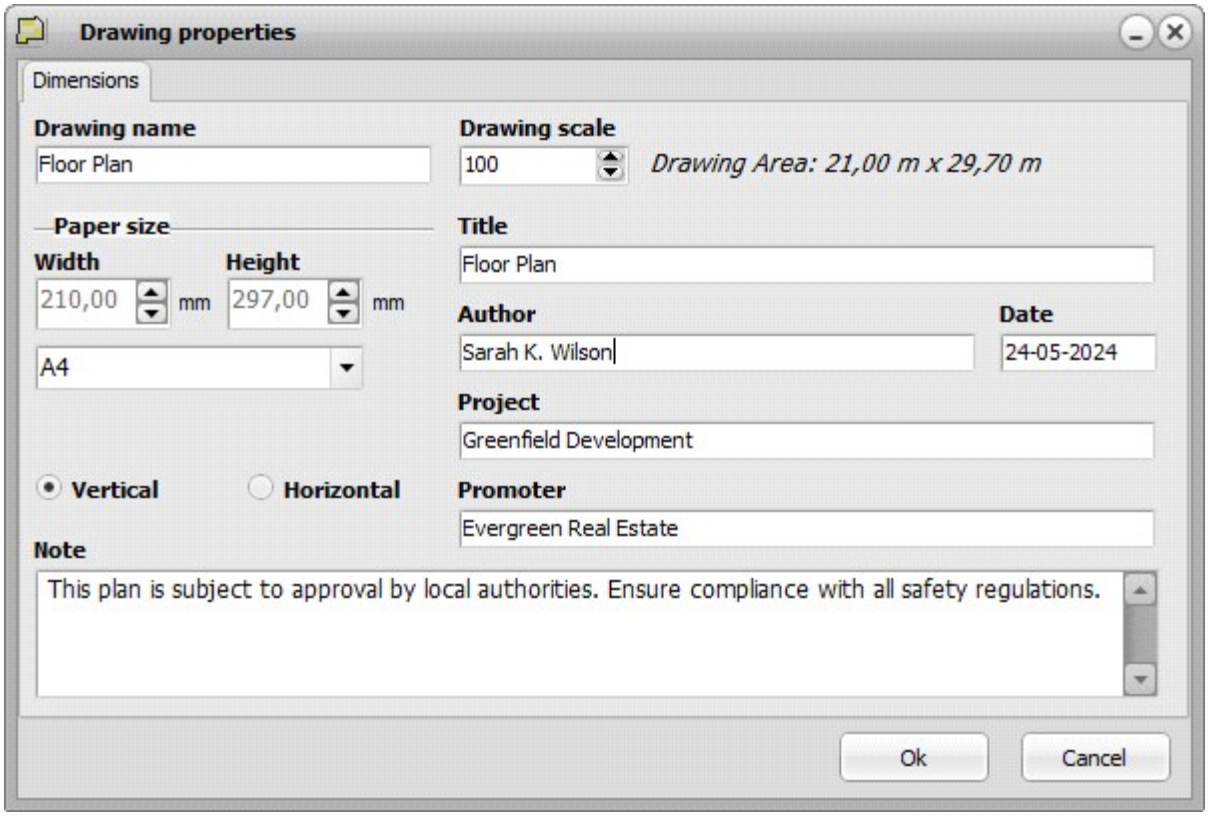


Fig.10 Properties editor for drawing and flow chart mode

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

We export 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.

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 into the editor from a stream, 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;
```

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.

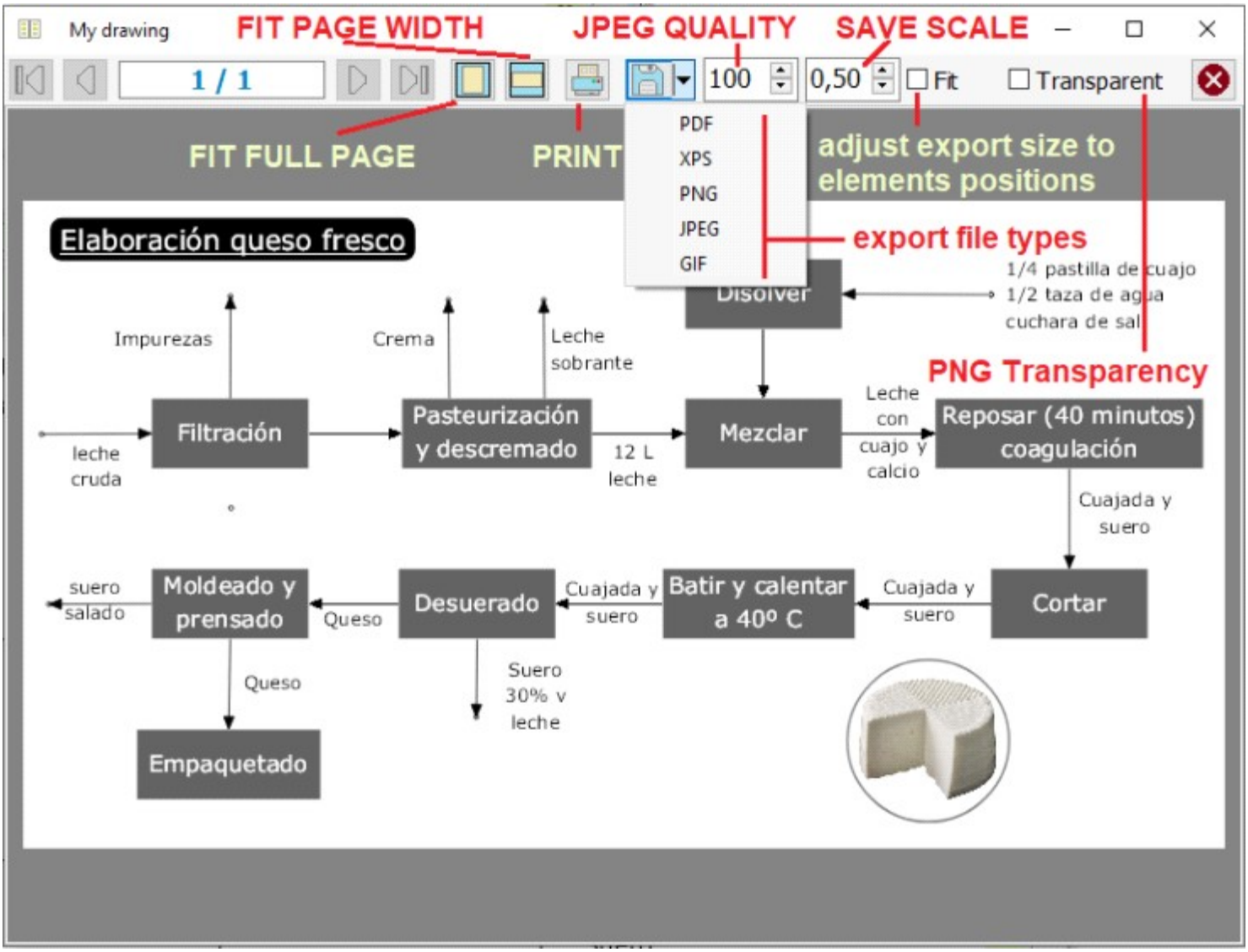


Fig.11 Preview

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. 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);

We save the content of the editor in 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;
```

5.4.3 Events

5.4.3.1 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 **TMPLabel** component. 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;
```

5.4.3.2 OnGetFontListEvent

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

With this event you can filter the text fonts to be displayed in the selection, 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;
```

5.5.TMPLInspector

The MPLInspector component is used in conjunction with the TMPLabel component 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 this 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.

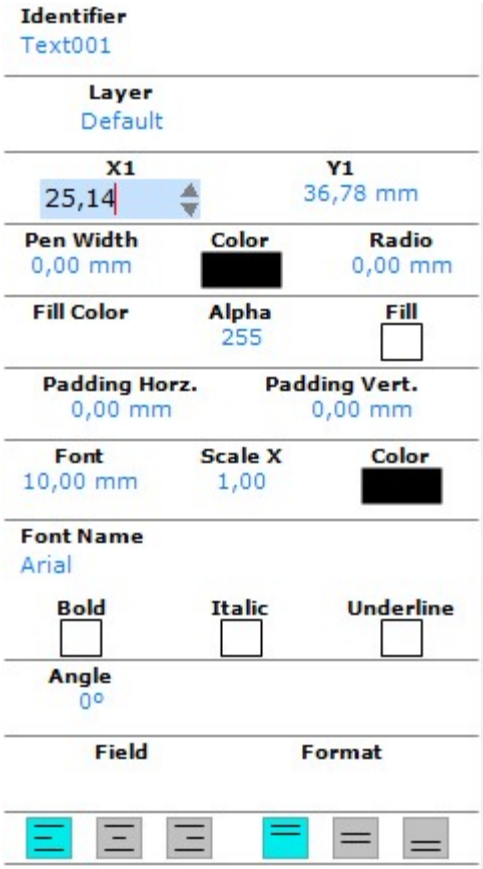


Fig.12 Inspector component

Property	Description
BackColor	TColor used as background of the control
EditColor	TColor used in text or number editors within the Inspector

Table2 TMPLInspector properties

5.6.TMPLEditor

5.6.4 Properties

5.6.4.3 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.13 Align label topleft

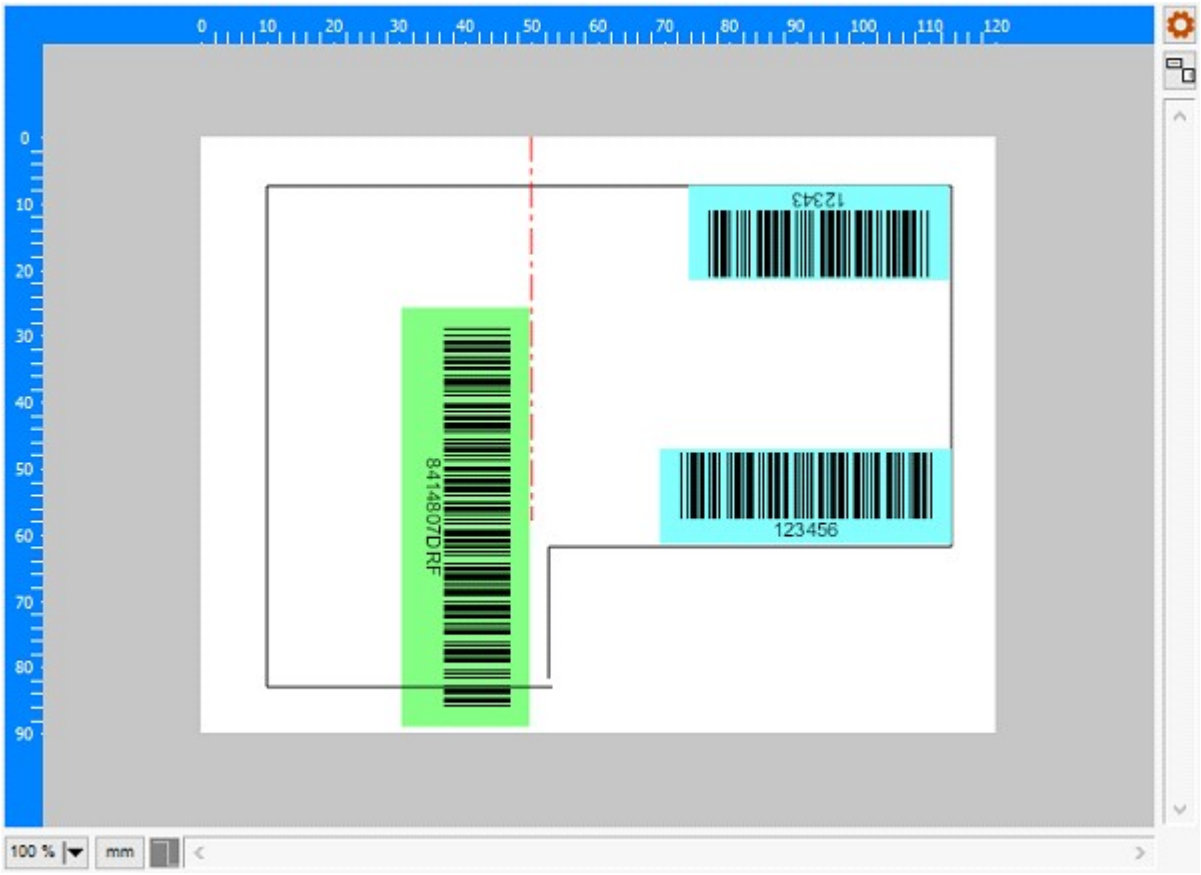


Fig.14 Align label centered

6.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.

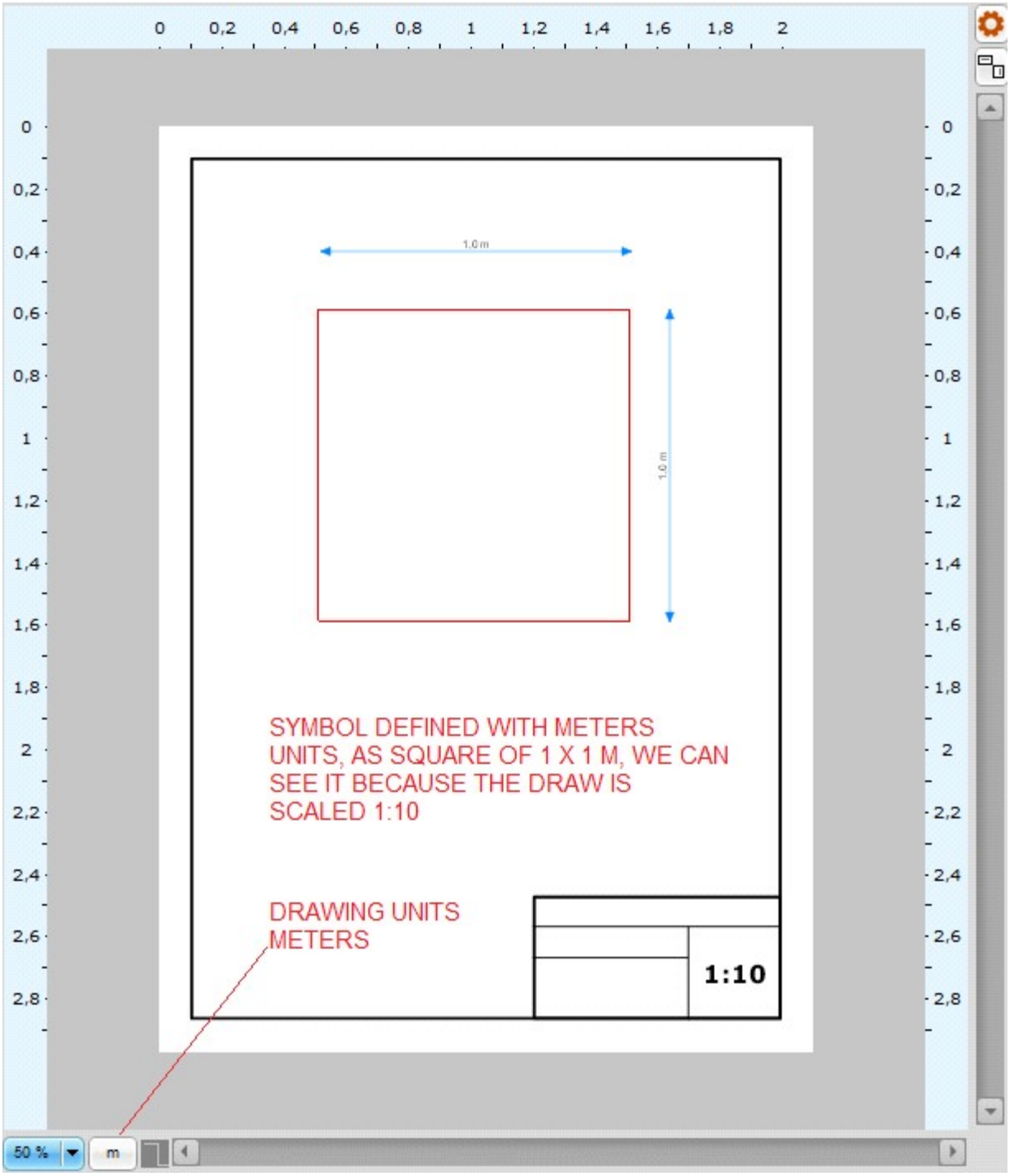


Fig.15 Symbol defined as square with 1 x 1 m

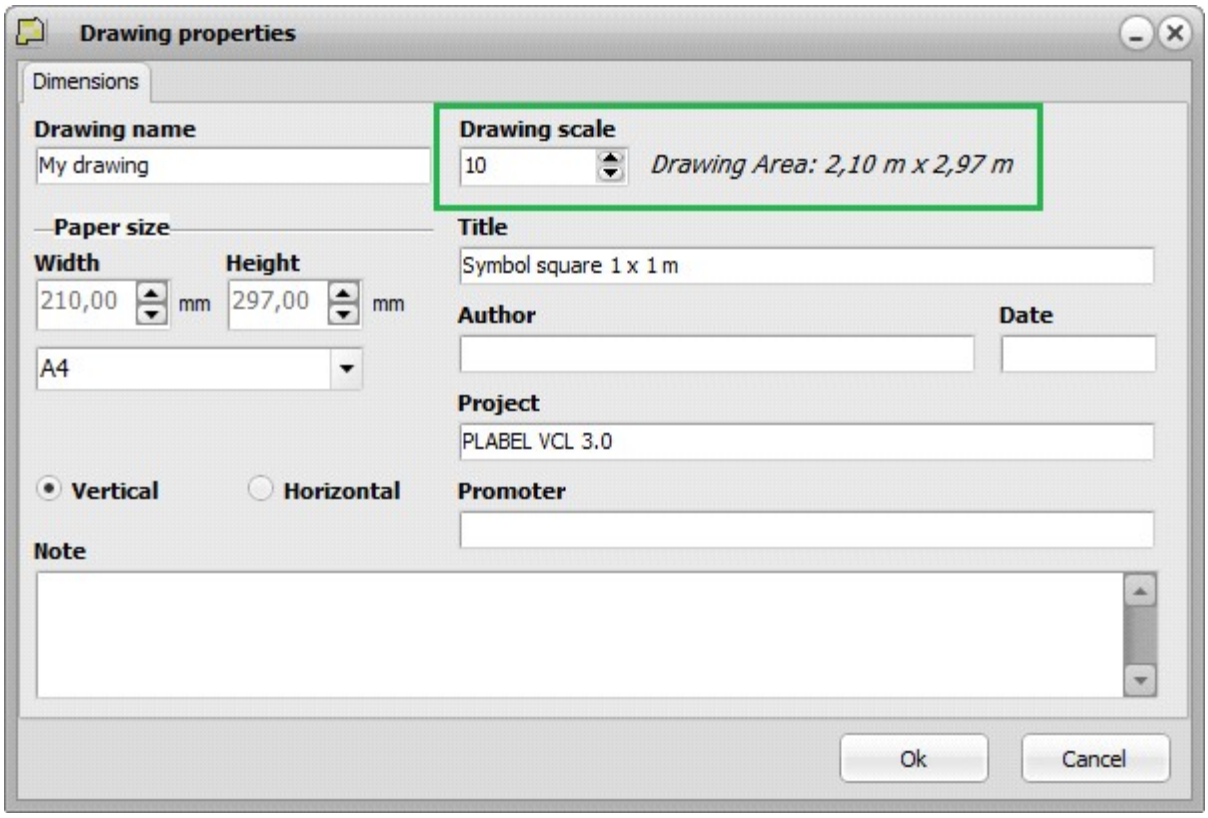


Fig.16 Drawing properties, setting 1:10 scale

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.

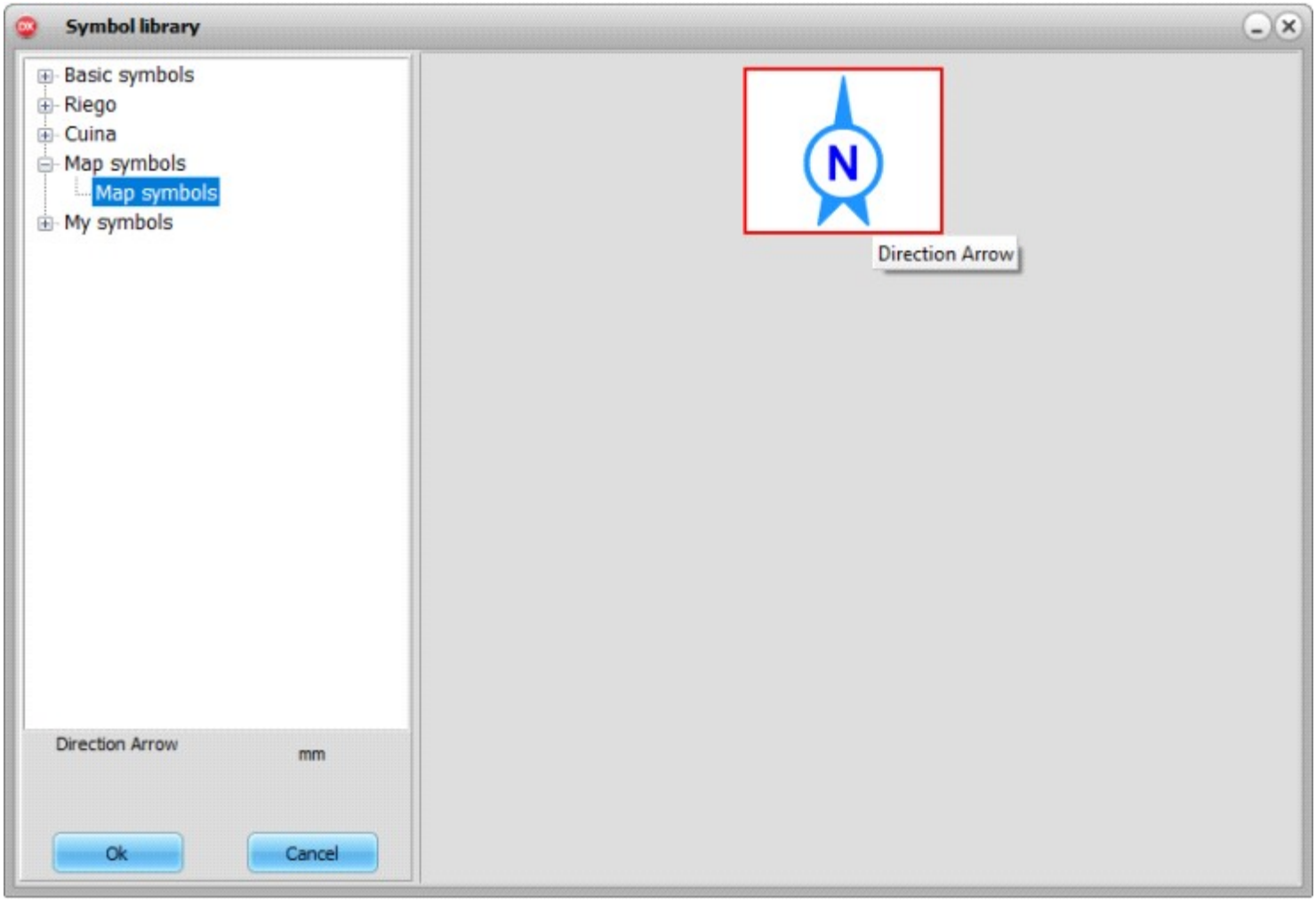


Fig.17 Map direction arrow

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.

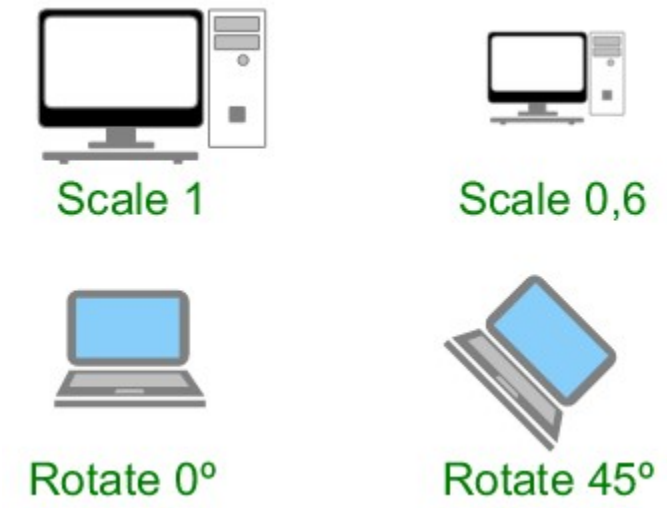


Fig.18 Scale and rotate symbol

7.List of figures

8.List of tables