EmbedDraw VCL & FMX STD

Developer Document

3.2.0

August 2026

1.Page index

2.Introduction 5
3.History 14
3.1 Version 3.0.0 - August 2024 14
3.2 Version 3.1.0 - January 2026 14
3.3 Version 3.2.0 - August 2026 14
4.Diagrams from Mermaid and PlantUML 15
5.GML maps and cadastral data 18
5.1 Supported coordinates and generated content 18
5.2.Parser APIs 20
5.3 GML parser 20
5.4 GPX parser 21
5.5 Mermaid and PlantUML parser model 22
6.Symbol path animations 24
6.1.TAnimationController and TPathAnimator 25
6.2 Path definitions 25
6.3 Controller methods 25
6.4 Timer, events and state 26
6.5 Animating along a drawn item 26
6.6 Supported item shapes 27
6.7 Low-level TPathAnimator 28
7.GPX tracks, routes and waypoints 29
7.1 Imported objects and map fitting 29
8.MCP Server 30
8.1 Visible updates after MCP tools 30
8.2.TEDWMCPServer public API 31
8.3 Lifecycle and extension methods 31
8.4 Connection and security properties 31
8.5 State and standard tools 32
8.6 Tool completion and repaint 32
9.AI tools 33
9.1 MCP Chat editor-view tool 33
9.2.TMCPAIClient public API 34
9.3 Response records and ownership 34
9.4 Configuration 34
10.Installation 35
10.1 Prerequisites 35
10.2 Automatic installation 35
10.3 Manual installation VCL 35
10.4 Manual installation FMX 38
10.5 Compatibility and optional services 40
11.Internationalization 41
12.Components 41
12.1.TEmbedDrawVCL - TEmbedDrawFMX 42
12.1.1 Properties 42
12.1.1.1 Action 42
12.1.1.2 BackSurf.Color 47
12.1.1.3 Data 47
12.1.1.4 DrawScale 48
12.1.1.5 EditorMode 48
12.1.1.6 Editable 48
12.1.1.7 ImageFolder 48
12.1.1.8 LanguageFileName 48
12.1.1.9 LayerZOrder 48
12.1.1.10 LibraryFolder 48
12.1.1.11 Mapping 48
12.1.1.12 Measure 49
12.1.1.13 OpenImgFolder 50
12.1.1.14 Options 50
12.1.1.15 Settings 51

12.1.1.16 Units 51
12.1.2 Methods 51
12.1.2.1 AddLink(const Id1, Id2: string; Idx1, Idx2: smallint; aType: TMPLinkType) 51
12.1.2.2 BeginUpdate 51
12.1.2.3 BringToFront 53
12.1.2.4 EditDimStyles 53
12.1.2.5 EditLayers 53
12.1.2.6 EditProperties 54
12.1.2.7 ExportDoc(aType: TMPLExport;const aFilename: TFileName;aPage: integer;aScale: single;aQuality: smallint;aCheckData,aTransparent: boolean;aFit: boolean = false) 56
12.1.2.8 FillBoxLayer(aReset: boolean = false) 56
12.1.2.9 LoadFromFile(const aFilename: TFileName); 57
12.1.2.10 LoadFromStream(aStream: TStream) 57
12.1.2.11 PasteSelection 58
12.1.2.12 Preview 58
12.1.2.13 Print 58
12.1.2.14 SameWidth 59
12.1.2.15 SameHeight 59
12.1.2.16 SaveToFile(const aFilename: TFileName;aFormat: TMPLExt = lfBin) 59
12.1.2.17 SaveSVG(const aFilename: TFileName;ExportType: TMPLExportSVG;Factor: single;attrID: boolean) 59
12.1.2.18 SaveToStream(aStream: TStream); 60
12.1.2.19 ElementByID 60
12.1.3 Events 60
12.1.3.1 OnBeforePrint 60
12.1.3.2 OnAfterPrint 60
12.1.3.3 OnGMLCoordinateTransform 61
12.1.3.4 OnInfoMsg 62
12.1.3.5 OnReadSymEvent 63
12.1.3.6 OnGetFontListEvent 63
12.1.3.7 OnUndoRedoEvent 63
12.1.3.8 OnGetListValues 64
12.1.3.9 OnSelectItem 64
12.1.3.10 OnUnSelectItem 64
12.2.TEmbedDraw public API 65
12.3 Document and serialization methods 65
12.4 Selection and arrangement 65
12.5 Layer methods 65
12.6 Read-only state and indexed collections 66
12.7.Document configuration objects 67
12.8 TEDWMeasure 67
12.9 TEDWLayer 67
12.10 TEDWMapping and related settings 68
12.11.TEDWData public API 69
12.12 Data-source properties 69
12.13 Reading values 69
12.14 CSV and runtime configuration 69
12.15 State properties 70
12.16.TEDWInspector 71
12.16.1 Properties 72
12.16.1.1 EmbedDraw 72
12.16.1.2 InspectPropBase 72
12.16.1.3 InspectProp 73
12.16.1.4 Options 73
12.16.1.5 BackColor 75
12.16.1.6 TitleColor 75
12.16.1.7 ValueColor 75
12.16.1.8 EditColor 75
12.16.1.9 SelColor 75
12.16.1.10 SelButColor 76
12.16.1.11 ArrowColor 76
12.17.TEDWEditor 77

12.17.1 Properties 77
12.17.1.1 EmbedDraw 77
12.17.1.2 Zoom 77
12.17.1.3 LabelPosition 77
12.17.1.4 Rulers 78
12.17.1.5 Grid 78
12.17.1.6 SelectionColors 79
12.17.1.7 MoveX 79
12.17.1.8 MoveY 79
12.17.1.9 Snap 79
12.17.1.10 OrtoLines 80
12.17.1.11 Margin 80
12.17.1.12 MarginColor 80
12.17.1.13 Gap 80
12.17.1.14 SizeRef 80
12.17.1.15 Label position 80
12.17.1.16 Options 81
12.17.1.17 EnabledMouseFilter 82
12.17.1.18 ShowRefLines 82
12.17.2 Methods 82
12.17.2.1 BeginUpdate 82
12.17.2.2 EndUpdate 82
12.17.2.3 EditSettings 83
12.17.2.4 AlignSelectedLeft 83
12.17.2.5 AlignSelectedRight 83
12.17.2.6 AlignSelectedTop 83
12.17.2.7 AlignSelectedBottom 83
12.17.2.8 ScrollToItem 84
12.17.3 Events 84
12.17.3.1 OnMouseDown 84
12.17.3.2 OnMouseUp 84
12.17.3.3 OnMouseMove 84
12.18.TEDWPreview 85
12.19.TEDWViewLibrary 86
12.20 Methods 86
12.21 Properties and event 86
12.22.TEDWSpinEditFloat 87
12.23.TEDWFactory 88
13.Create Symbols 90
13.1 Available symbol primitives 94
13.2 Style and construction instructions 94
13.3 Useful definition, instance and library members 95
14.List of figures 97
15.List of tables 98

2.Introduction

The **EmbedDraw VCL and FMX STD** packages provides components for label design and printing, flow charts, editable diagrams and CAD-style technical drawings in Delphi VCL and FireMonkey applications. It supports 32-bit and 64-bit targets from Delphi 10 Seattle through Delphi 13 Florence. Version 3.2 adopts the new `TEmbedDrawVCL`, `TEmbedDrawFMX`, `TEDWInspector` and `TEDWEditor` component names and adds Mermaid /PlantUML parsing to include in flow chart mode, symbol path animation, GML and GPX map loading, an MCP server and optional AI tools.

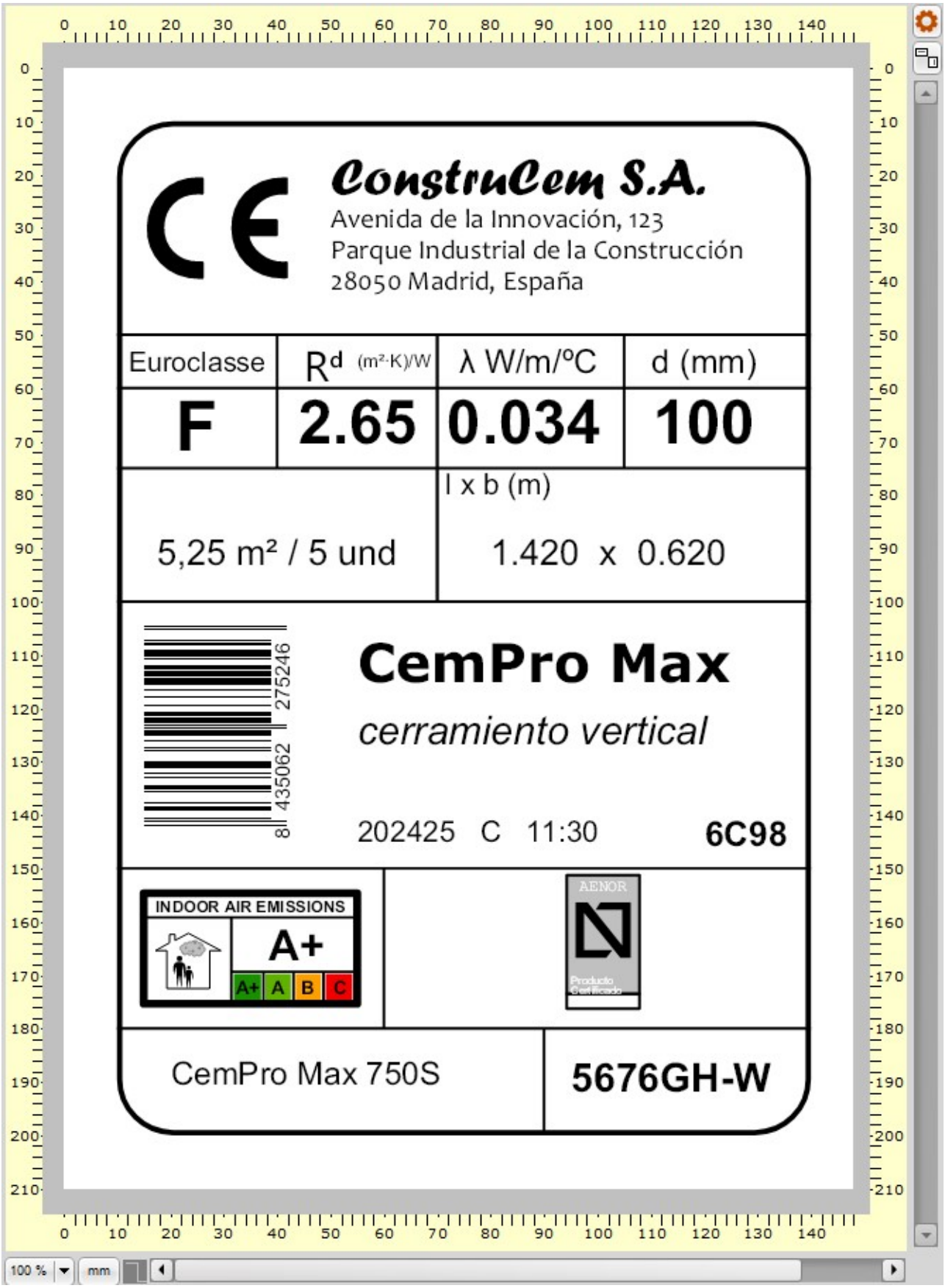


Fig.1 Label editor

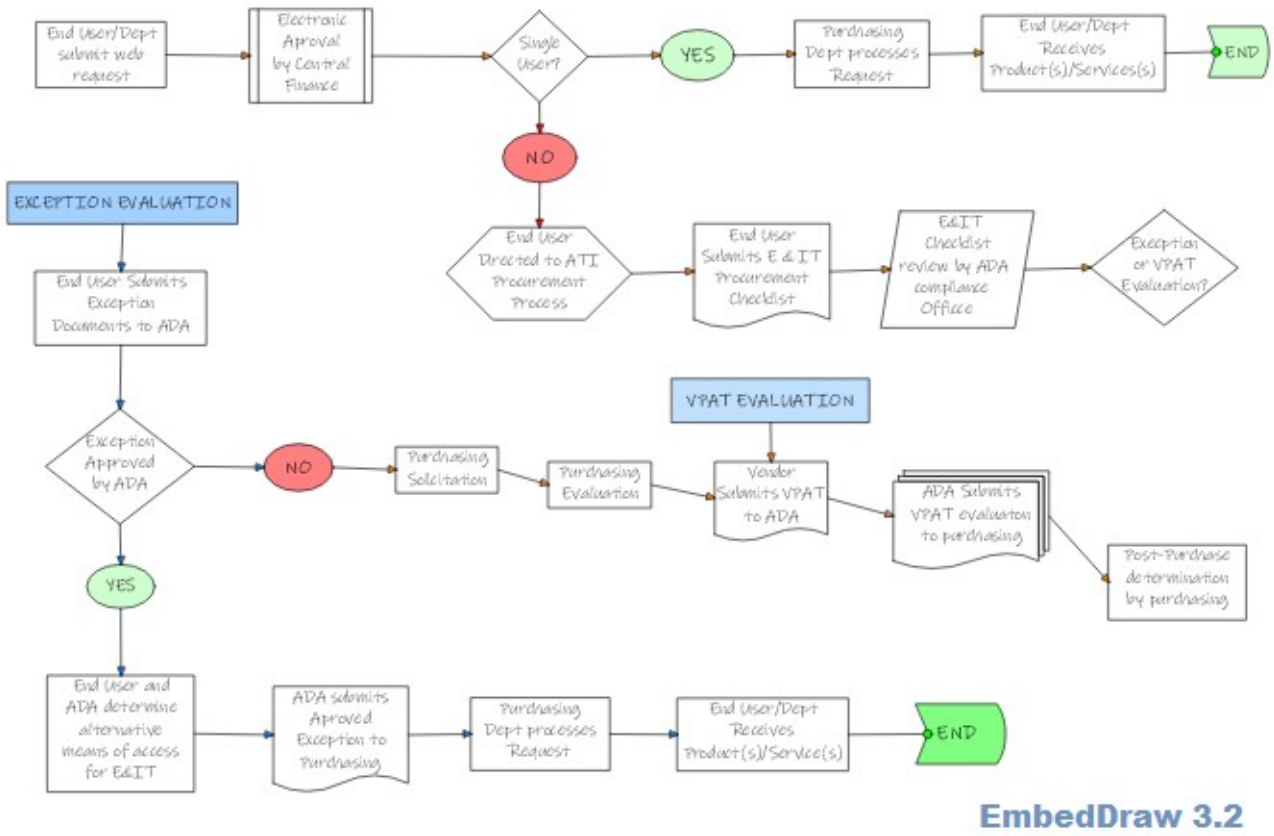


Fig.2 Flow chart

Fig.3 Technical drawing example

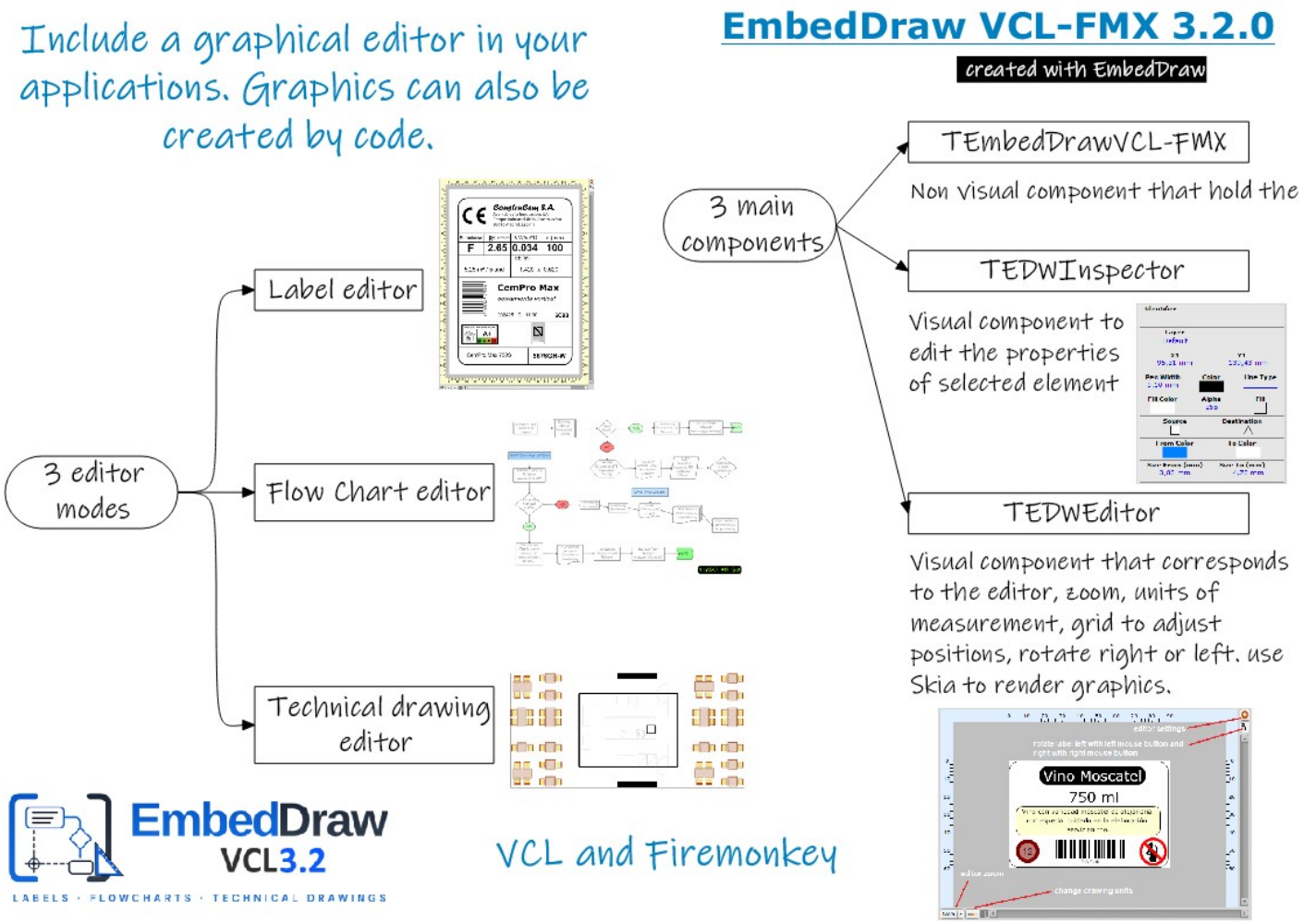


Fig.4 EmbedDraw VCL-FMX 3.2.0

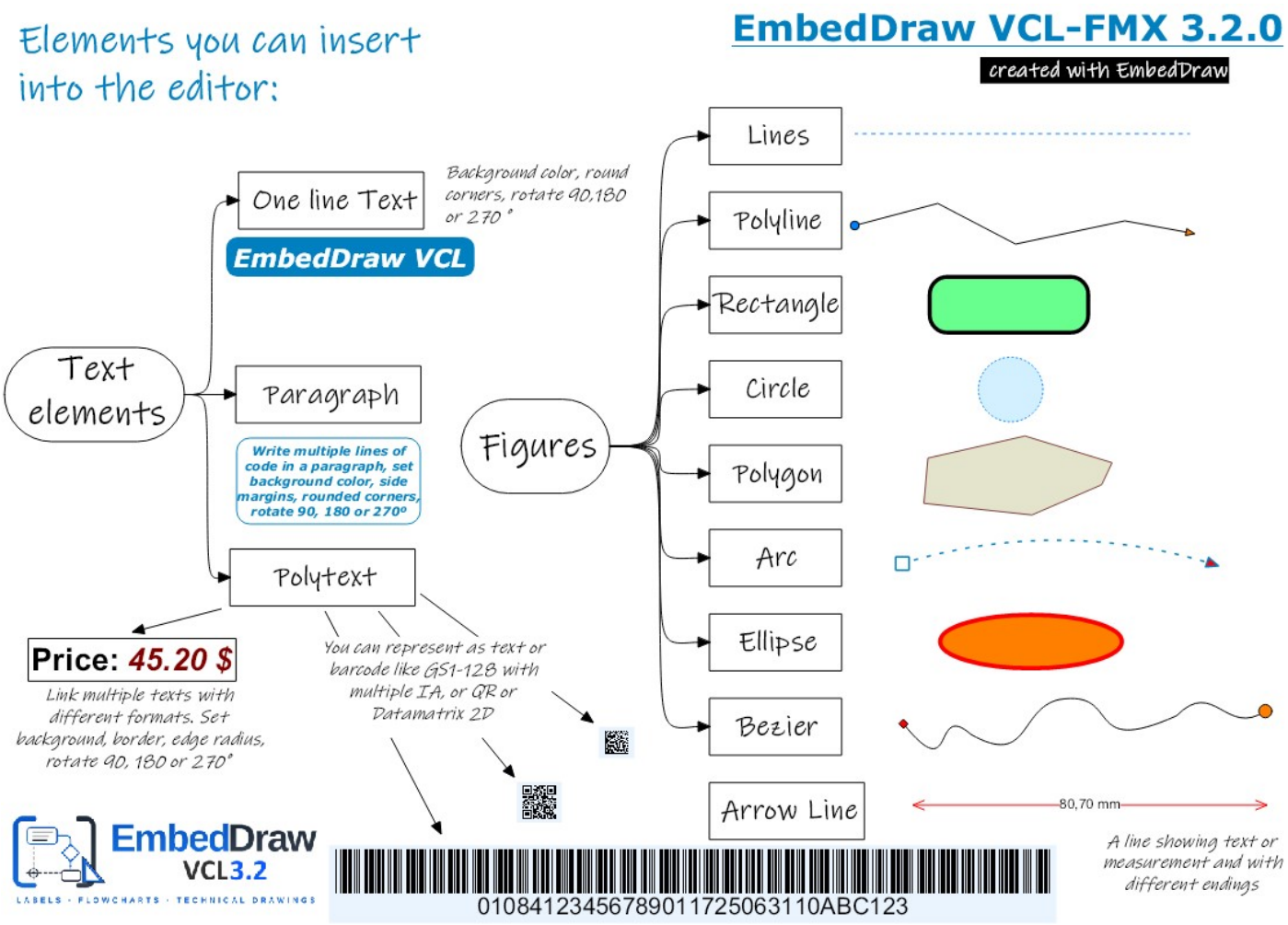


Fig.5 Elements you can insert

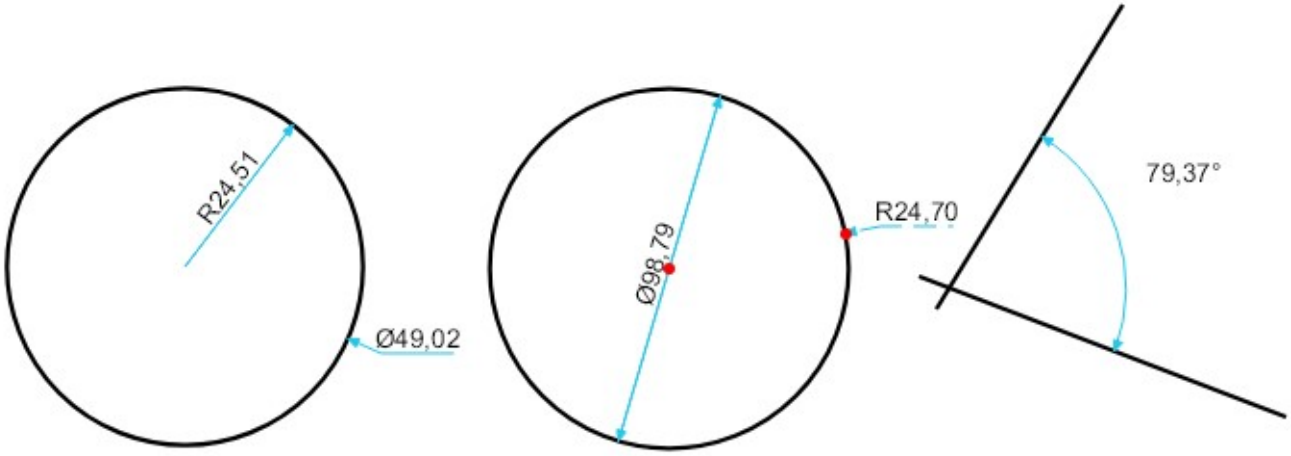


Fig.6 New in version 3.1 radial, diameter and angle dimenion styles.

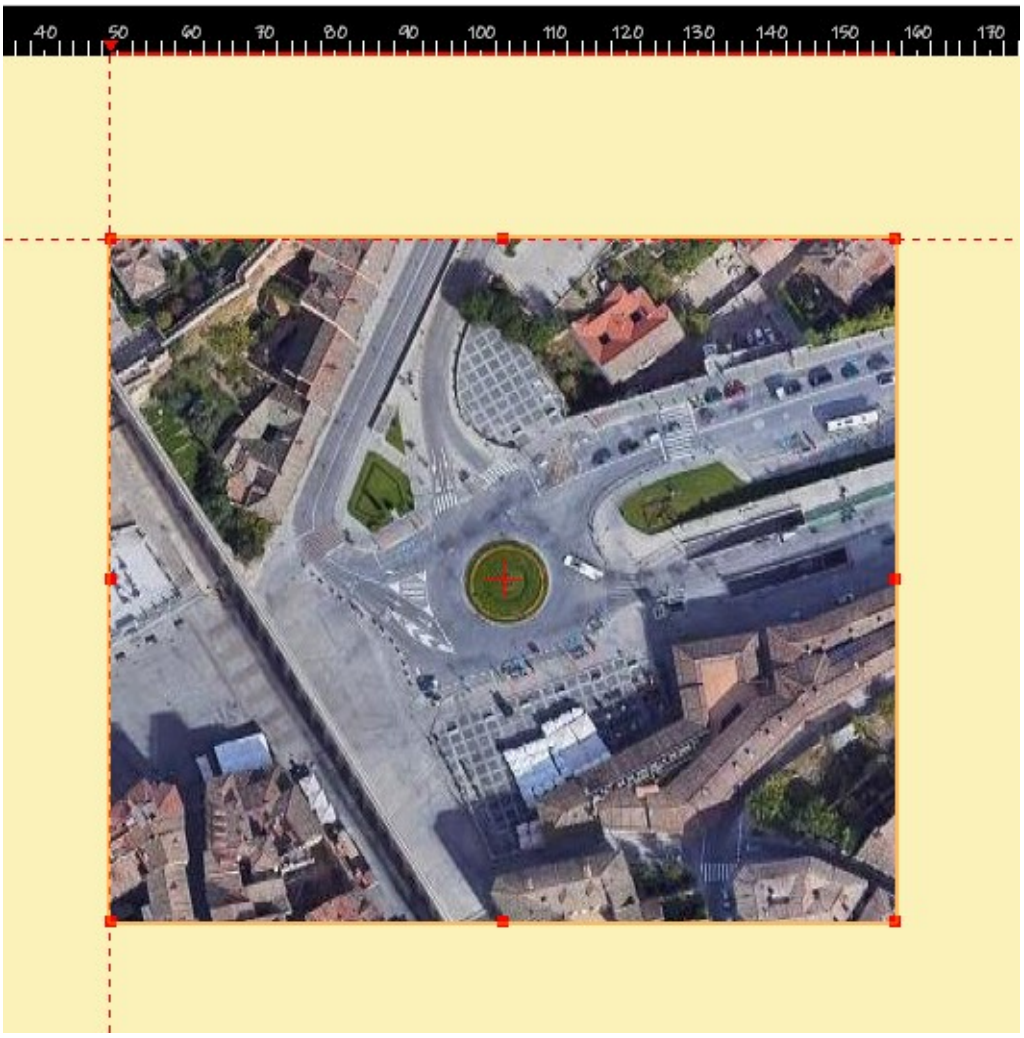


Fig.7 New in version 3.1 Image of map services in the image element or as the background of a layer.

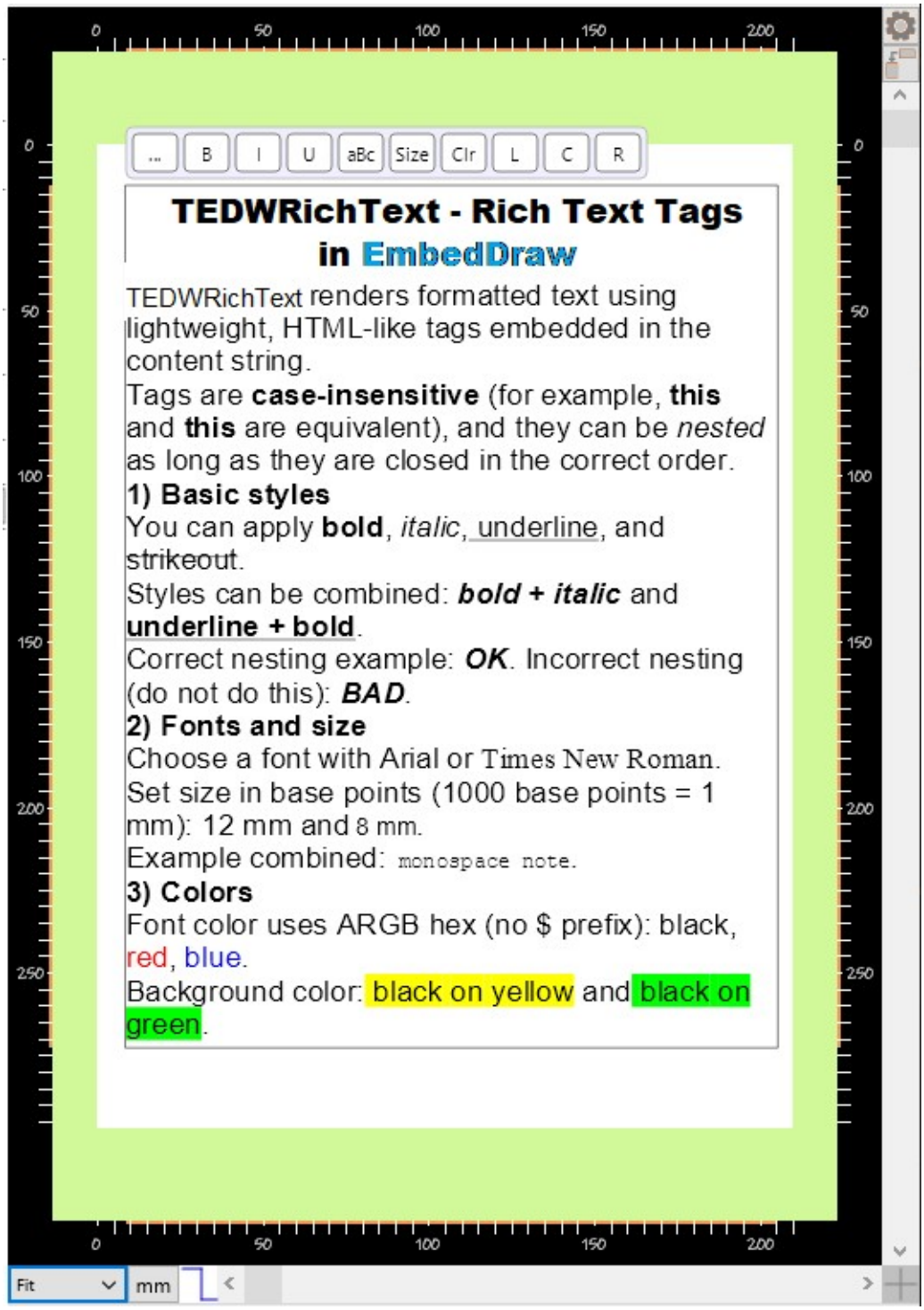


Fig.8 New in version 3.1: TEDWRichText element to show formated text.

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 EmbedDraw 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

3.3 Version 3.2.0 - August 2026

- Renamed the main components and classes to the EmbedDraw/TEDW naming scheme.
- Added Mermaid and PlantUML parsing to generate editable diagrams and flow charts.
- Added symbol animation through geometric paths.
- Added loading and display of GML cadastral polygons and GPX tracks, routes and waypoints over map layers.
- Added the TEDWMCPServer component and drawing tools for Model Context Protocol clients.
- Added optional Anthropic, OpenAI and Gemini AI tool integration, including symbol-library generation from images.

4.Diagrams from Mermaid and PlantUML

EmbedDraw 3.2 imports textual Mermaid and PlantUML descriptions and converts them into native, editable EmbedDraw items. Nodes become flow-chart shapes, relationships become connectors, and labels remain text objects; the result can therefore be edited, printed and exported like any other drawing.

Component API

```
function LoadFromDiagram(const aFilename: TFileName;
  aFormat: TMPLDiagramFormat = dgAuto): Boolean;

function LoadFromDiagramText(const aText: string;
  aFormat: TMPLDiagramFormat = dgAuto): Boolean;

EmbedDraw1.LoadFromDiagram('C:\Diagrams\order.mmd', dgMermaid);
EmbedDraw1.LoadFromDiagram('C:\Diagrams\login.puml', dgPlantUML);
EmbedDraw1.LoadFromDiagram('C:\Diagrams\diagram.txt'); // dgAuto
EmbedDraw1.LoadFromDiagramText(Source, dgAuto);
```

`TMPLDiagramFormat` contains `dgAuto` , `dgMermaid` and `dgPlantUML` . With `dgAuto` , files ending in `.mmd` or `.mermaid` are treated as Mermaid; `.puml` , `.plantuml` , `.pu` , `.iuml` and `.wsd` are treated as PlantUML. If the extension is inconclusive, the source is inspected; for example, `@startuml` identifies PlantUML. Files are read as UTF-8 first and fall back to ANSI.

Load behavior

- A successful import replaces the current document content with native elements.
- The method returns `True` when at least one item was created.
- The document units are set to millimetres and the paper size is adjusted to the generated content.
- `emFlowChart` is the recommended editor mode for imported diagrams.
- Unsupported statements can be skipped so that a useful partial diagram is still produced. Use the parser diagnostics, or strict parser mode, when validation must reject partial input.

Supported Mermaid input

Family	Supported content
Flow chart / graph	`TD` , `TB` , `BT` , `LR` and `RL` headers; process, terminator, predefined-process, database, start/end, decision, preparation, data, manual-operation and manual-input node shapes; solid, dotted, labelled, chained and grouped connections.
Sequence	Participants and actors with aliases, messages with the common solid/dotted arrow variants, and one-line notes placed left, right or over participants.
Gantt	Basic title, sections, start dates, durations and milestones.

Additional 3.2 families	State, class, mind-map, entity/relationship, requirements, git graph, timeline, journey, C4 and the other structured families recognised by `TEDWMermaidParser`.

For flow charts, Mermaid node syntax is mapped to `TMPLFlowType`. Examples include `A[Process]`, `A{Decision}`, `A[(Database)]`, `A((Start))`, `A(((End)))`, `A[/Data/]` and `A>Manual input]`. Connection examples include `A --> B`, `A -.-> B`, `A -->|label| B`, `A --x B`, `A --o B`, `A --> B --> C` and `A & B --> C`.

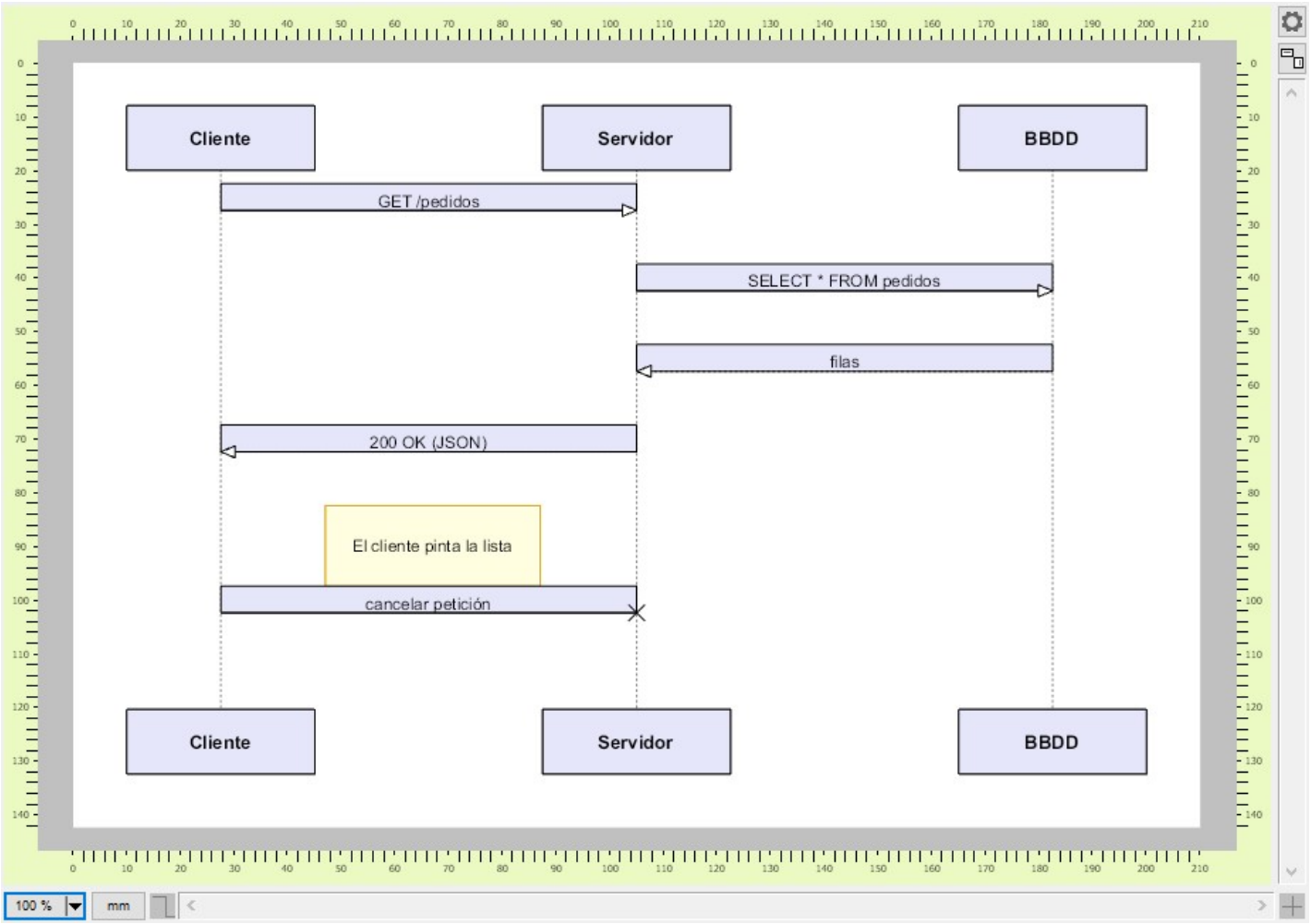


Fig.9 Mermaid Sequence diagram

```
sequenceDiagram
  participant C as Cliente
  participant S as Servidor
  participant B as BBDD
  C->>S: GET /pedidos
  S->>B: SELECT * FROM pedidos
  B-->>S: filas
  S-->>C: 200 OK (JSON)
  Note right of C: El cliente pinta la lista
  C-xS: cancelar petición
```

Supported PlantUML input

Family	Supported content
Activity	`start` , `stop` / `end` , action statements, `if/else/endif` branches and `while` /`endwhile` loops.
Sequence and state	Participants, actors and common participant types, messages and one-line notes; state declarations, aliases, labelled transitions and `[*]` initial/final states.
Gantt and mind map	Tasks, dependencies, dates, progress, milestones and colours; OrgMode star-depth and arithmetic plus/minus mind-map hierarchies.
Archimate	The standard local `Rel_*` relationship macros, including triggering, serving, realization, influence, flow, composition, association, assignment, aggregation and access variants. No external include is downloaded.
Additional 3.2 families	Structural UML, WBS, C4, timing, network, chronology, JSON/YAML data, Salt, grammar, file-tree, Ditaa and Chen diagrams recognised by `TEDWPlantUMLParser` .

Architecture and direct parser API

`MPL.DiagramParsers` is framework-neutral RTL code. It parses either language into a `TEDWDiagramModel` ; the VCL and FMX factories render that model with the same layout rules. `MPL.Base` reaches the platform factory through the loader registered with `MPLSetDiagramLoader` .

```
Model:= MPLParseDiagram(Source, dgMermaid);
 try
 // Inspect Model.Kind, Nodes, Edges, Participants and Diagnostics.
 finally
  Model.Free;
 end;
```

The three-argument overload `MPLParseDiagram(Source, Format, Strict)` raises `EMPLDiagramParseError` on the first warning or error when `Strict` is `True` . Each diagnostic supplies severity, source line, code, message and source text. At factory level, `TEDWFactory.LoadDiagram` is the common entry point; `LoadMermaid` remains as a compatibility wrapper. The MCP tool `load_diagram` accepts `code` , `format` (`auto` , `mermaid` or `plantuml`) and optional strict validation.

Rendering notes and current limitations

Flow charts are laid out by levels, sequence diagrams use participants, lifelines and vertically ordered events, and backward connections are routed around nodes. Generated labels are separate native text items. Some language-specific presentation is intentionally simplified: unsupported styles, subgraph containers, complex sequence frames, activation bars, compound states and advanced activity constructs may be ignored or represented as a simpler editable diagram. `BT` and `RL` currently use the corresponding layout axis without reversing declaration order.

5.GML maps and cadastral data

`LoadFromGML` reads cadastral polygon geometry, converts the declared coordinate reference system to geographic coordinates, creates a map layer and a parcel layer, and composes an online map beneath the geometry.

```
EmbedDraw1.LoadFromGML('parcel.gml', 19, mpESRI,
  gmtLayerBackground);
```

Fig.10 Load GML files

The loader supports CRS84, EPSG:4326, 4258, 4269, 4283 and 3857, together with common northern and southern UTM families. `OnGMLCoordinateTransform` can handle an additional CRS. The map request is asynchronous; the provider, zoom, centre and exportable background are stored in the map layer when composition completes.

5.1 Supported coordinates and generated content

The parser requires an EPSG declaration and polygon `posList` coordinates. Native conversion covers WGS84-style geographic systems EPSG:4326, 4258, 4269 and 4283; Web Mercator EPSG:3857; northern UTM families

258xx, 269xx and 326xx; southern UTM families 283xx and 327xx; and GDA2020 MGA zones EPSG:7846-7859. Use `OnGMLCoordinateTransform` for another CRS and set `Handled` only after producing valid latitude and longitude.

Import creates a non-selectable **GML map** layer and a selectable **GML parcels** layer. The requested provider and zoom are stored on the map layer and added to `Mapping.MapProviders`. `gmtLayerBackground` includes the map in background SVG/export output; the alternative target keeps it as normal layer content. Map composition is asynchronous, and another GML/GPX import cannot begin until it finishes.

5.5.Parser APIs

EmbedDraw exposes framework-neutral parsers when an application needs to inspect or validate input before creating drawing elements. These units belong to the shared RTL code and do not depend on VCL or FMX.

5.3 GML parser

Member/type	Description
`TEDWGMLParser.ParseFile(FileName, out Data)`	Reads UTF-8, falling back to ANSI, and parses polygon `posList` elements.
`TEDWGMLParser.ParseText(Text, out Data)`	Parses GML already held in memory.
`TMPLGMLData`	Returns `EPSG`, an array of rings and MinX/MinY/MaxX /MaxY bounds. Each ring contains `TMPLGMLPoint` X/Y coordinates.

The parser removes a duplicated closing point because EmbedDraw polygons close themselves. It raises `EFileNotFoundException` for a missing file and `EConvertError` for absent CRS declarations, invalid coordinates or files without usable polygon rings.

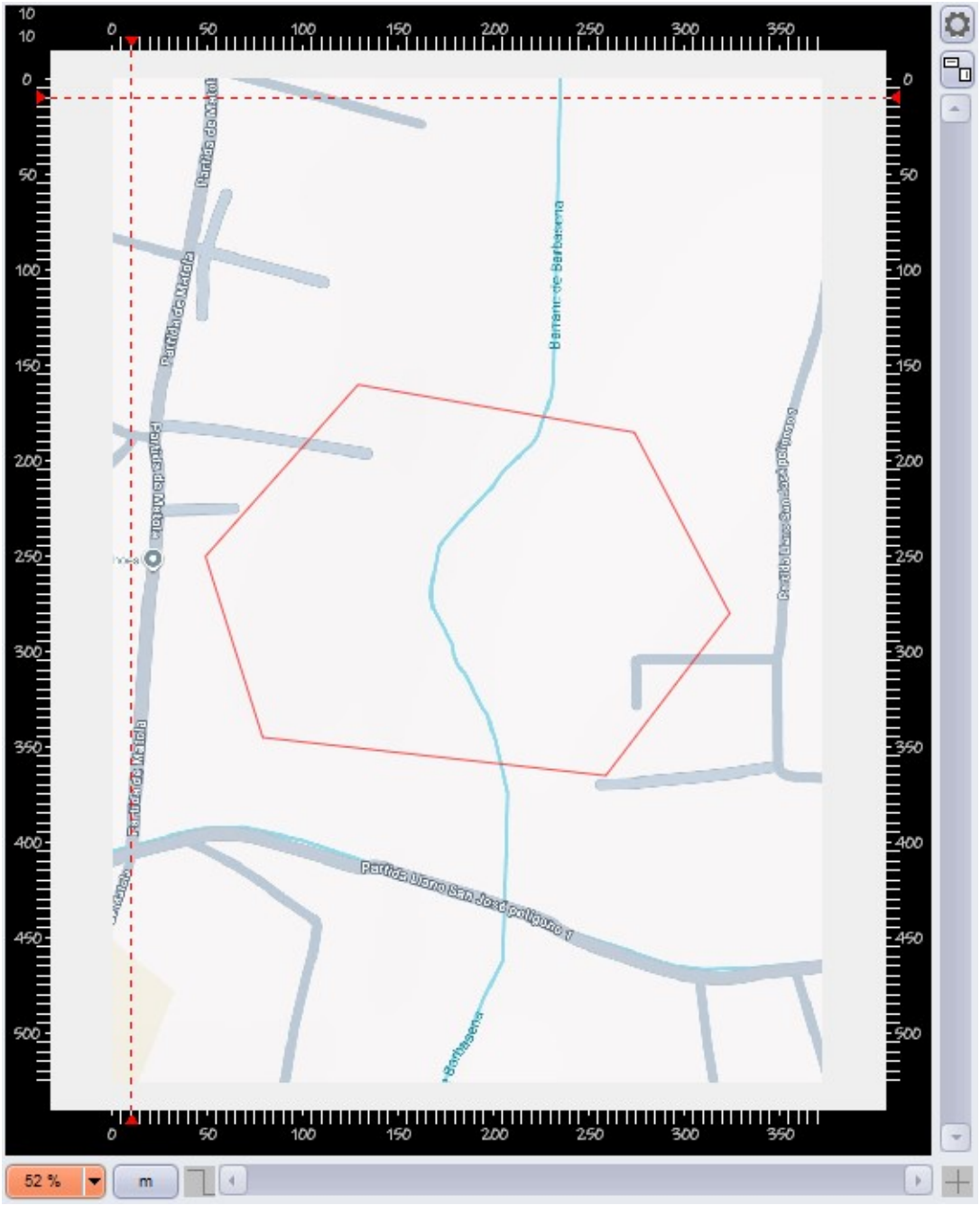


Fig.11 Load GML files

5.4 GPX parser

Member/type	Description
`TEDWGPXParser.ParseFile(FileName, out Data)`	Reads and parses a GPX file.
`TEDWGPXParser.ParseText(Text, out Data)`	Parses an in-memory GPX document.
`TMPLGPXData`	Returns paths, waypoints and geographic bounds.
`TMPLGPXPath`	Contains track/route kind, name and points.

`TMPLGPXPoint`	Contains latitude, longitude, optional elevation, name and timestamp text.

Routes or track segments with fewer than two valid points are not emitted as paths. Invalid latitude/longitude values and documents without tracks, routes or waypoints raise `EConvertError`.

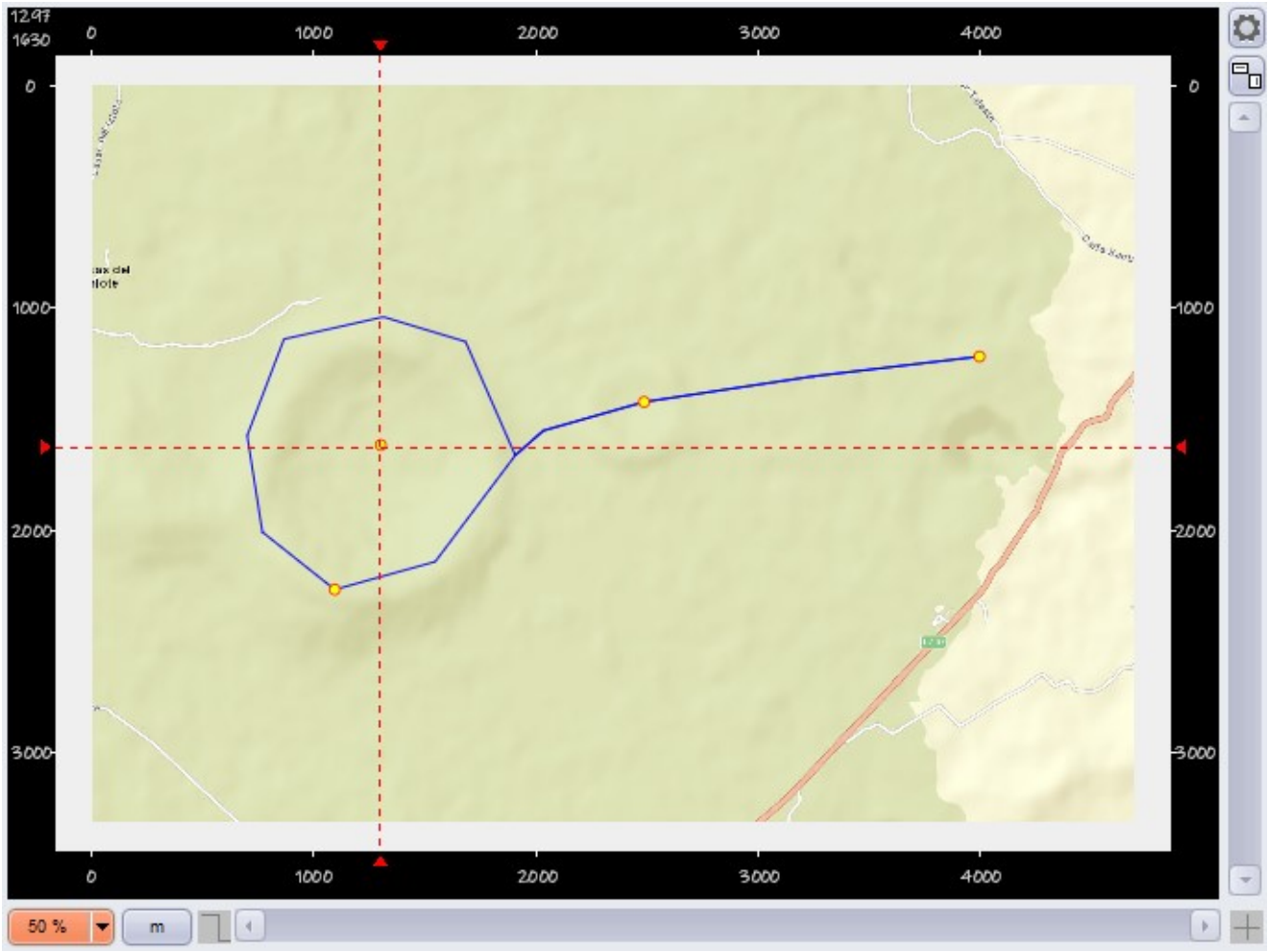


Fig.12 Load GPX files

5.5 Mermaid and PlantUML parser model

Member	Description
`MPLParseDiagram(Text, Format[, Strict])`	Returns a newly allocated `TEDWDiagramModel`; the caller owns and must free it. Strict mode raises `EMPLDiagramParseError` on the first warning or error.
`MPLDetectDiagramFormat`	Detects Mermaid or PlantUML from text.
`MPLDiagramFormatFromExt`	Detects format from supported filename extensions and returns `dgAuto`

`TEDWDiagramModel`	Owns nodes, edges, participants, sequence events and structured diagnostics. Public builders include `AddNode`, `AddEdge`, `AddParticipant`, `AddSeqEvent` and `AddDiagnostic`.
`HasDiagnostics(MinimumSeverity)`	Tests for informational messages, warnings or errors at or above the requested severity.

```
Model := MPLParseDiagram(Source, dgAuto, True);
try
  // Inspect Model.Nodes, Model.Edges and Model.Diagnostics.
finally
  Model.Free;
end;
```

6.Symbol path animations

Version 3.2 adds path animation for `TEDWSymbol` objects. `TAnimationController`, implemented for VCL and FMX, moves one or more symbols through line, polyline, circle, ellipse, arc or bezier paths. Animation requires `EditorMode = emInteractive`.

The controller supports start, pause, stop, reset and normalized seeking; individual and global time scales; start delays; repetition; ping-pong movement; automatic rotation along the tangent; easing curves; and completion, cycle and waypoint callbacks.

```
Id := AnimationController.AddSymbol(Symbol, ptPolyline, Segments, 5.0);
AnimationController.ConfigureSymbol(Id, True, 0, 0, True);
AnimationController.SetSymbolEasing(Id, peEaseInOut);
AnimationController.StartSymbol(Id);
```

The lower-level `TPathAnimator` class can also be used when an application wants to drive updates manually.

A path does not have to be built segment by segment. `TAnimationController` can also take the trajectory from a figure already drawn in the document, deducing whether it is a line, polyline, polygon, rectangle, circle, ellipse or bezier, and animating the symbol over it with a single call that only needs both item identifiers and the duration. See *TAnimationController and TPathAnimator*.

6.7.TAnimationController and TPathAnimator

TAnimationController is available in `MPL.VCL.AnimationController` and `MPL.FMX.AnimationController`. It owns the timing loop and coordinates one or more `TEDWSymbol` instances. Interactive movement requires the owner drawing to use `EditorMode = emInteractive`.

6.2 Path definitions

Type	Values or fields	Purpose
`TPathType`	ptLine, ptPolyline, ptCircle, ptEllipse, ptArc, ptBezier	Selects the geometric interpretation of the path segments.
`TPathEasing`	peLinear, peEaseIn, peEaseOut, peEaseInOut, peSmoothStep	Controls acceleration over the complete normalized path.
`TPathSegment`	StartPoint, EndPoint, ControlPoint1 /2, Radius, RadiusY, StartAngle, EndAngle, SpeedFactor	Describes a segment. A SpeedFactor of 0 or 1 is normal speed, 0.5 is half speed and 2 is double speed.

6.3 Controller methods

Member	Description
`AddSymbol(...): Integer`	Registers a symbol and returns the identifier used by all subsequent operations.
`AddSymbolRepeated(...)`	Registers a repeated animation using simulated duration and a total repeat count; zero repeats indefinitely.
`ConfigureSymbol(ID, AutoRotate, AngleOffset, StartDelay, PingPong)`	Configures orientation, delayed start and reverse travel.
`SetSymbolEvents`	Assigns cycle and waypoint callbacks. `AddSymbol` also accepts a completion callback.

`StartSymbol`, `PauseSymbol`, `StopSymbol`, `ResetSymbol`, `RemoveSymbol`	Control the lifetime of a single registered symbol.
`StartAll`, `PauseAll`, `StopAll`, `ResetAll`	Apply the corresponding operation to the complete controller.
`SeekSymbol(ID, Progress)`	Moves immediately to a normalized progress value from 0 to 1.
`WaitSymbol(ID, SimulatedSeconds)`	Inserts a simulated wait into the symbol schedule.
`SetSymbolTiming`, `SetSymbolEasing`, `SetGlobalTimeScale`, `SetSimulationScale`	Configure per-symbol and global timing.
`Update(DeltaTime)`	Advances animations manually when the internal timer is not used.

6.4 Timer, events and state

Member	Description
`StartTimer` / `StopTimer`	Enable or disable the internal UI timer.
`UpdateInterval`	Timer interval in milliseconds.
`MaximumDeltaTime`	Caps elapsed time after a blocked UI frame to prevent an animation jump.
`Running`, `GlobalTimeScale`, `GetActiveSymbolCount`	Expose controller status.
`GetSymbolState`, `IsSymbolActive`, `GetSymbolProgress`, `GetSymbolPathProgress`, `GetSymbolEasing`	Query individual animation state, time progress and geometric progress.
`OnBeforeUpdate` / `OnAfterUpdate`	Run application logic immediately before and after each controller update.

6.5 Animating along a drawn item

The controller can take the animation path directly from an item already present in the drawing, deducing the path type from the item shape. This avoids building the `TPathSegment` array by hand when the trajectory is a figure the user has drawn or the application has created with `TEDWFactory`. Coordinates are read in the document current units.

Member	Description
`AnimateItemByID(Owner, SymbolID, PathID, SimulatedDuration, Loop, AngleOffset): Integer`	Resolves both the symbol and the path item by their document ID, deduces the shape, configures automatic rotation, places the symbol at the origin and starts the internal timer. `SimulatedDuration` is the time in seconds for one complete travel of the figure. Returns the animation identifier. Raises `EInvalidOperation` when either identifier is not found.
`AddSymbolOnItem(Symbol, PathItem, SimulatedDuration, RepeatCount, OnComplete): Integer`	Same shape deduction when the symbol and the item are already available as objects. Registers the animation without starting it, so `ConfigureSymbol` and `SetSymbolEvents` can still be applied. A repeat count of zero repeats indefinitely.
`PathFromItem(PathItem, out PathType, out Segments): Boolean`	Class function that performs only the conversion. Returns False when the item shape cannot be used as a path. Useful to adjust `SpeedFactor` per segment before calling `AddSymbolRepeated`.

6.6 Supported item shapes

Item	Resulting TPathType	Segments
Line, arrow line	`ptLine`	One segment between both points.
Polyline	`ptPolyline`	One segment per pair of consecutive points.
Polygon	`ptPolyline`	Closed: a final segment returns to the first vertex.
Rectangle, square	`ptPolyline`	Four segments along the perimeter, starting at the first corner.
Circle	`ptCircle`	Centre and radius taken from the bounding box. One complete turn.
Ellipse	`ptEllipse`	Centre, `Radius` and `RadiusY` taken from the bounding box. One complete turn.
Bezier	`ptBezier`	One cubic segment per group of three points, matching the 3N+1 point layout of the item.

Any other item, including arcs, sectors, text and tables, is rejected: `PathFromItem` returns False and `AddSymbolOnItem` raises `EInvalidOperation`. The drawing must use `EditorMode = emInteractive`, as with any other animation.

```
// The controller is kept in a form field so the animation can be paused,
// stopped or queried later. It is owned by the form and freed with it.
var
 IdA: integer;
 AnimationController: TAnimationController;
begin
 if MPLabel1.EditorMode <> emInteractive then
   MPLabel1.EditorMode := emInteractive;
```

```
 AnimationController := TAnimationController.Create(Self);
 AnimationController.OnBeforeUpdate := AnimBeforeUpdate;
 AnimationController.OnAfterUpdate := AnimAfterUpdate;

 // Symbol ID, drawn item ID and seconds for one complete travel.
 // The shape of Fig01 is deduced and the internal timer is started.
 IdA := AnimationController.AnimateItemByID(MPLabel1, 'Sym01', 'Fig01', 8.0);
end;

// Both handlers bracket the frame so the drawing is repainted once,
// with every animated symbol already in its new position.
procedure TForm1.AnimBeforeUpdate(Sender: TObject);
begin
 MPLabel1.BeginUpdate;
end;

procedure TForm1.AnimAfterUpdate(Sender: TObject);
begin
 MPLabel1.EndUpdate;
end;
```

Variations are applied on the returned identifier with the usual members. `AnimateItemByID` loops by default; pass `Loop` as False for a single pass, and `AngleOffset` as -90 when the symbol is drawn pointing up instead of to the right. Once running, `PauseSymbol`, `StopSymbol`, `SetSymbolEasing` and `SetSymbolEvents` accept that same identifier.

6.7 Low-level TPathAnimator

`TPathAnimator` can be used without a symbol controller. Its public API includes `SetPath`, `SetTiming`, `CalculatePosition`, `Update`, `Reset`, `Seek`, reverse and easing control, path length and crossed-waypoint queries. The controller is recommended unless the application needs to apply the calculated X, Y and rotation values itself.

7.GPX tracks, routes and waypoints

`LoadFromGPX` reads GPX 1.0/1.1 files in WGS84 and creates editable polylines for tracks and routes plus marker circles for waypoints. Separate `trkseg` elements remain separate paths, so discontinuous recordings are not joined accidentally.

```
EmbedDraw1.LoadFromGPX('route.gpx', 0, mpOpenStreetMap,
  gmtLayerBackground);
```

A zoom value of zero automatically fits all GPX content. Tracks are blue, routes are red and waypoints are yellow. Names, timestamps and elevations are parsed and the map uses the same asynchronous layer workflow as GML.

7.1 Imported objects and map fitting

GPX 1.x waypoints (`wpt`), routes (`rte/rtept`) and track segments (`trk/trkseg/trkpt`) are supported, including names, elevation and time metadata at parser level. Tracks become blue polylines, routes red polylines and waypoints red/yellow circle markers on the selectable **GPX tracks** layer. The online image is stored on a separate **GPX map** layer.

A zoom of zero automatically fits the complete geographic extent with padding; an explicit zoom must be within the map subsystem's supported range. The selected provider, calculated center and final zoom are stored on the map layer and in `Mapping`. As with GML, map generation is asynchronous and `ImageFolder` defaults to the GPX file directory when it was empty.

8.MCP Server

`TEDWMCPServer` exposes an EmbedDraw component through the Model Context Protocol over Streamable HTTP. Assign `DrawLabel`, configure `Port` and `EndpointPath` (default `/mcp`), then set `Active` or call `Start`.

```
MCPServer.DrawLabel := EmbedDraw1;
MCPServer.Port := 3001;
MCPServer.Active := True;
```

The default VCL and FMX tool sets let an MCP client inspect and configure a drawing, manage layers, create and modify all principal element types, load diagrams and Gantt data, work with symbol libraries, render previews, and save or export SVG and other document formats.

The server binds to localhost by default. Configure `AuthToken`, `BindAddress`, `AllowedOrigins`, payload limits and logging deliberately before exposing it to another machine.

8.1 Visible updates after MCP tools

After a tool that adds, changes, deletes, clears, loads or creates drawing content completes, `TEDWMCPServer` flushes pending geometry calculations and sends a paint notification before returning the JSON-RPC result. MCP clients therefore do not need a separate refresh call: new text, lines and other figures are immediately visible in every attached editor.

8.6.TEDWMCPServer public API

TEDWMCPServer is a non-visual component declared in `MPL.MCP`. The VCL package registers it on the EmbedDraw palette; FMX applications can create it at run time. Platform tool units register the standard drawing tools automatically during initialization.

8.3 Lifecycle and extension methods

Member	Description
`Start` / `Stop`	Starts or stops Streamable HTTP. Setting `Active` calls the same lifecycle methods.
`RegisterDefaultTools`	Registers the platform's standard EmbedDraw tool set when the list is empty.
`RegisterTool(Name, Description, InputSchema, Handler)`	Adds an application-specific MCP tool. Ownership of `InputSchema` passes to the server and it is released with the server.
`BuildAIToolDefinitions`	Builds provider-neutral tool definitions for direct AI calls.
`EffectiveInstructions`	Returns custom `Instructions`, or the built-in EmbedDraw workflow when empty.
`Log`	Sends a diagnostic message through `OnLog`.

8.4 Connection and security properties

Property	Default	Description
`DrawLabel`	nil	Drawing exposed to MCP tools.
`Port`	3001	HTTP listener port.
`BindAddress`	127.0.0.1	Listener address. A non-loopback address requires a non-empty `AuthToken`.
`EndpointPath`	/mcp	Single Streamable HTTP endpoint.
`AllowedOrigins`	Empty	Semicolon-separated CORS origins. Empty accepts only local origins.
`AuthToken`	Empty	Bearer token. It is public rather than published so it is not stored in form files.
`MaxRequestBytes`	8,388,608	Maximum JSON request size.
`MaxAIImageBytes`	20,971,520	Maximum decoded image accepted by AI image tools.
`ToolListCacheTTL`	300000 ms	Freshness hint included in modern tools/list responses.
`LogPayloads`	False	Logs full JSON payloads. Enable only when drawing and prompt data may safely be written to logs.
`OnLog`	nil	Application diagnostic callback.

8.5 State and standard tools

`Initialized`, `ProtocolVersion`, `SessionId` and `Tools` expose current server state. Version 3.2 registers 59 standard tools covering update transactions, document inspection, page and map setup, items, layers, geometric and textual elements, codes, images, tables, symbols, dimensions, diagrams, Gantt, preview, SVG, document export and native save/load.

8.6 Tool completion and repaint

The server automatically completes pending calculations and repaints `DrawLabel` after mutating tools. A successful tool result therefore means both the data model and attached editors have been updated. The 59-tool count refers only to the platform standard set; applications may use `RegisterTool` to add tools such as `set_editor_view` in the MCP Chat demo.

9.AI tools

The MCP component also provides direct AI integration for Anthropic, OpenAI and Gemini through `TMCPAIProvider`. Configure `AIProvider`, `AIModel`, `AIMaxTokens` and the non-published `AIApiKey` property at run time; API keys are intentionally not stored in form files.

`BuildAIToolDefinitions` exposes provider-neutral tool definitions, allowing an application to run an AI tool loop against the same drawing operations offered by MCP. The `create_symbol_library_from_image` tool can analyse an input image, generate reusable vector-style symbol commands and save a `.lyp` library. `MaxAIImageBytes` limits decoded image input.

AI output should be reviewed before it is saved or used in production drawings, and credentials must be supplied from secure application configuration.

9.1 MCP Chat editor-view tool

The VCL `Demos\VCL\MCP\MCPChat` application registers the application-specific `set_editor_view` tool in addition to the 59 standard drawing tools, so this demo exposes 60 tools. It changes the attached `TEDWEditor` viewport and does not modify the drawing or printed/exported output.

Argument	Values	Description
`zoom`	`25`, `50`, `75`, `100`, `125`, `150`, `175`, `200`, `250`, `300`, `350`, `400`, `450`, `500`, `width`, `height`, `fit_all`	Maps to the corresponding `TEDWEditor.Zoom` value.
`label_position`	`center`, `top_left`	Maps to `TEDWEditor.LabelPosition` values `lbCenter` and `lbTopLeft`.

```
{"zoom":"150","label_position":"center"}
```

Both arguments are optional, allowing the model to change only zoom or only page placement. The demo initially uses `zmFitAll` and `lbCenter`.

9.4.TMCPAIClient public API

TMCPAIClient, declared in `MPL.MCP.AI`, adapts the same EmbedDraw tool definitions to Anthropic, OpenAI and Gemini request formats. Construct it with the configured `TEDWMCPServer`; the server supplies provider, model, API key, token limit and tool definitions.

Member	Description
`Create(AServer)`	Links the client to an MCP server configuration. The server must outlive the client.
`Send(Messages, SystemPrompt, Tools): TAIResponse`	Sends a provider-native HTTP request and returns text, tool calls, stop reason and error information.
`MakeUserMessage`	Builds a text user message.
`MakeUserImageMessage`	Builds a user message containing base64 image data and its media type.
`MakeAssistantMessage`	Builds a normal assistant response for conversation history.
`MakeAssistantToolUse` / `MakeAssistantToolUses`	Build assistant messages containing one or several requested tool calls.
`MakeToolResult`	Builds the message returned to the model after executing a requested tool.

9.3 Response records and ownership

`TAIResponse` contains `Text`, `ToolCalls`, `StopReason`, `HasToolUse` and `ErrorMessage`. Each `TAIToolCall` contains `Id`, `Name`, `ThoughtSignature` and parsed `Arguments`. Do not free an Arguments object: it is owned by the client and remains valid only until the next `Send` call or client destruction.

9.4 Configuration

Server property	Description
`AIProvider`	`aiAnthropic`, `aiOpenAI` or `aiGemini`.
`AIModel`	Provider model identifier. The shipped default is `claude-sonnet-4-20250514`.
`AIMaxTokens`	Maximum generated tokens; default 4096.
`AIApiKey`	Runtime secret. It is deliberately not published and must not be persisted in DFM /FMX form resources.

The host application owns the tool loop: after `Send` reports tool calls, validate each call, execute the corresponding controlled operation, append tool-result messages and call `Send` again until no further tool use is requested.

10.Installation

10.1 Prerequisites

EmbedDraw 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 EmbedDraw, 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 EmbedDraw VCL and FMX STD components can be done in two ways:

10.2 Automatic installation

It is the recommended way, run the EmbedDraw 3.2 installer supplied for your framework, Delphi compiler version and license. If your distribution does not include an installer, use the manual procedure below. 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.

10.3 Manual installation VCL

If you need to recompile or reinstall the components, go to the folder where the group file is located, the MPL.EmbedDraw.groupproj project group has three packages: MPL.EmbedDraw.RTL (common code for VCL and FMX), MPL.EmbedDraw.VCL (VCL components) and MPL.EmbedDraw.DSGN.VCL (component functionalities for design within IDE). Open the group from the package folder matching the compiler version (D100, D101, D102, D103, D104, D110, D120 or D130).

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 **TEDWInspector** , **TEDWViewLibrary** , **TEDWEditor** and **TEDWSpinEditFloat** components, and finally when opening the third project run Build and Install which will install the **TEmbedDrawVCL** component in the EmbedDraw VCL STD tab.

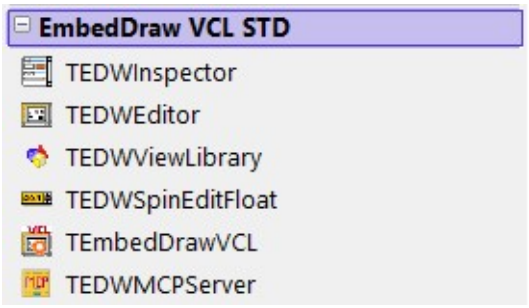


Fig.13 EmbedDraw VCL components

TEDWSpinEditFloat	Descendant of TCustomEdit that is used in various forms for entering and formatting decimal values.
TEDWEditor	Visual component for designing and editing labels and diagrams, connects to the TEmbedDrawVCL base component
TEDWInspector	Visual component to view and modify the properties of selected elements in the editor, connects to the TEmbedDrawVCL base component
TEDWViewLibrary	Used to display a library of *TEDWSymbol* elements in selection dialog.
TEmbedDrawVCL	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 EmbedDraw 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\EmbedDraw\VCL\3_2\Source;
- C:\Users\Public\Documents\Mesurasoft\EmbedDraw\VCL\3_2\Source\VCL;
- C:\Users\Public\Documents\Mesurasoft\EmbedDraw\VCL\3_2\Source\Shared

B)- Define a variable (for example *$(EMBEDDRAW_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.

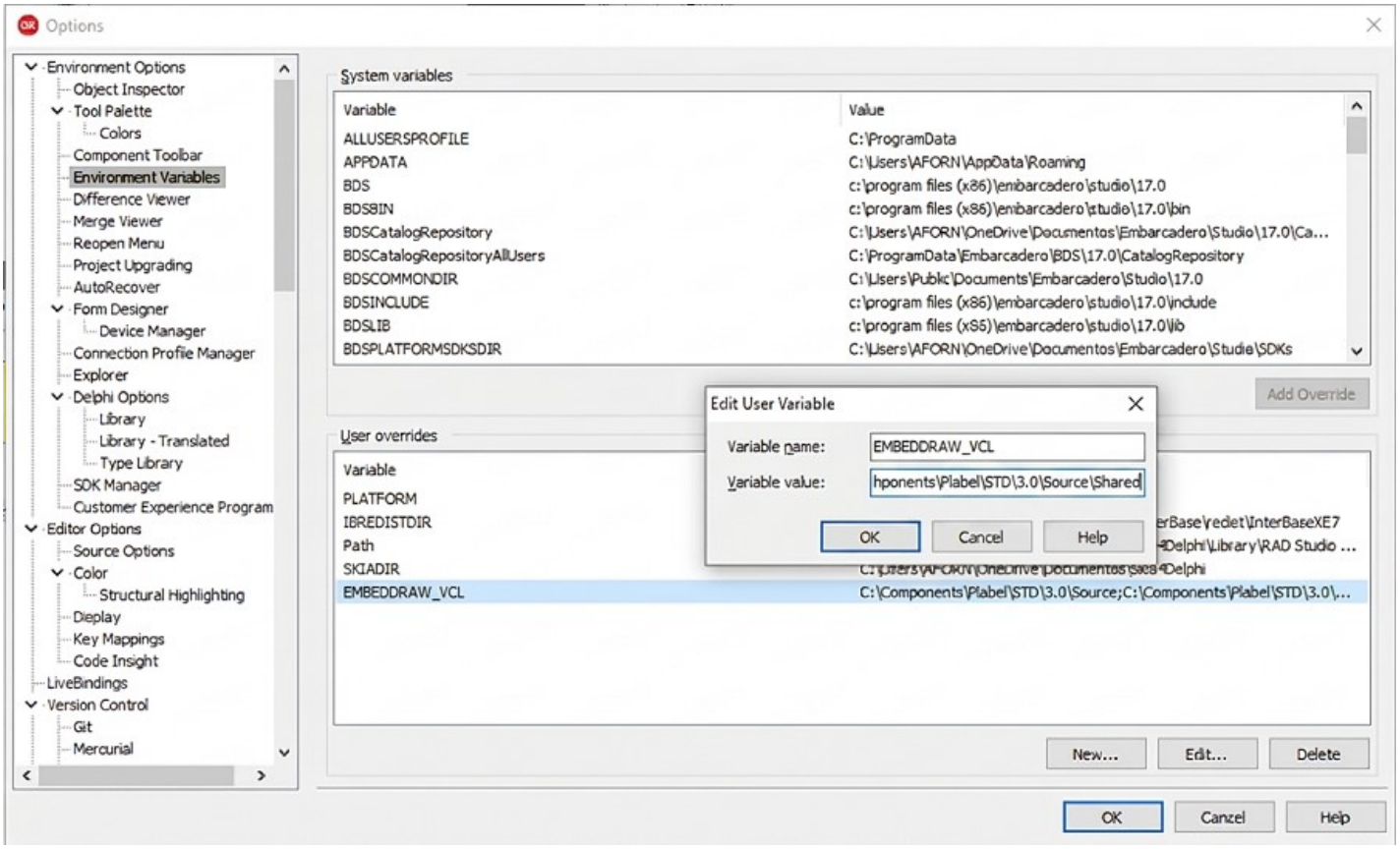


Fig.14 Define new variable

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

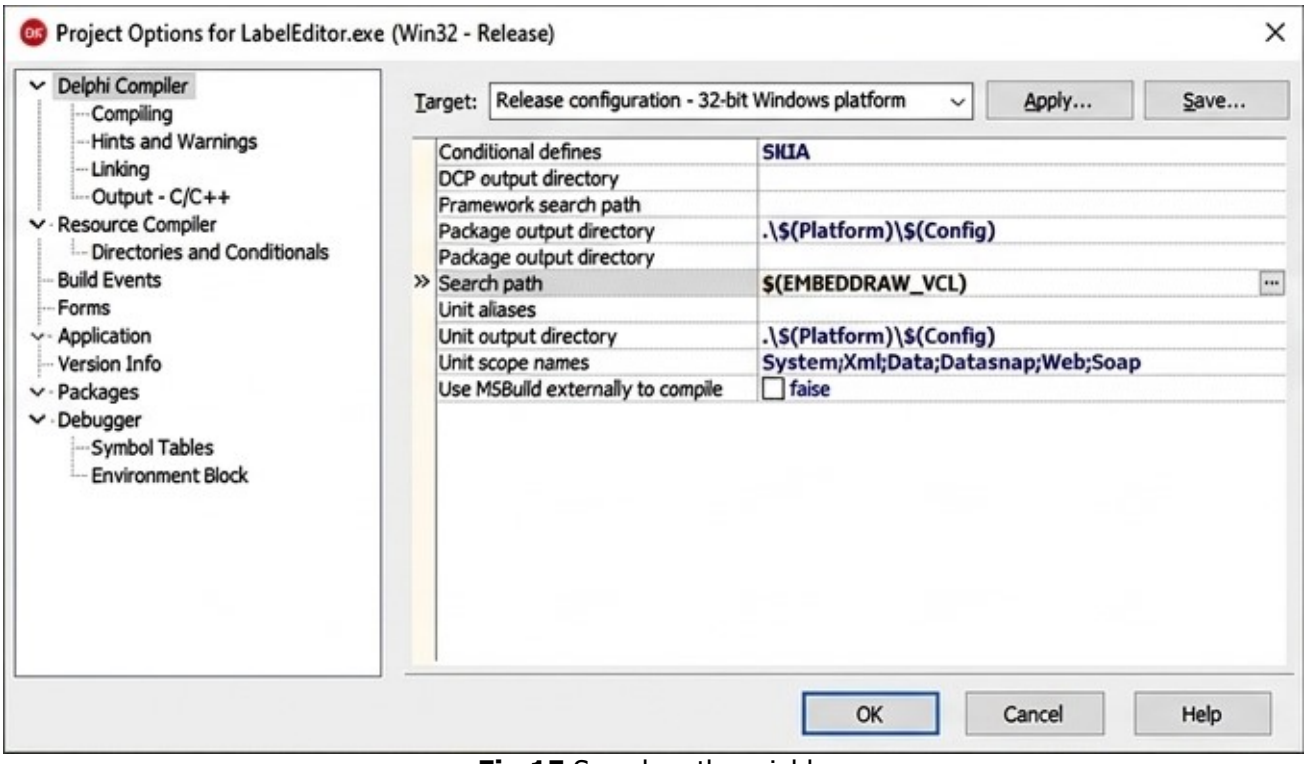


Fig.15 Search path variable

10.4 Manual installation FMX

If you need to recompile or reinstall the components, go to the folder where the group file is located, the MPL.EmbedDraw.groupproj project group has three packages: MPL.EmbedDraw.RTL (common code for VCL and FMX), MPL.EmbedDraw.FMX (FMX components) and MPL.EmbedDraw.DSGN.FMX (component functionalities for design within IDE). Open the group from the package folder matching the compiler version (D100, D101, D102, D103, D104, D110, D120 or D130).

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 **TEDWInspector** , **TEDWViewLibrary** and **TEDWEditor** components, and finally when opening the third project run Build and Install which will install the **TEmbedDrawFMX** component in the EmbedDraw FMX STD tab.

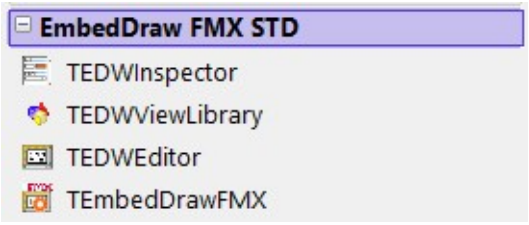


Fig.16 EmbedDraw FMX Components

TEDWEditor	Visual component for designing and editing labels and diagrams, connects to the TEmbedDrawFMX base component
TEDWInspector	Visual component to view and modify the properties of selected elements in the editor, connects to the TEmbedDrawFMX base component
TEDWViewLibrary	Used to display a library of *TEDWSymbol* elements in selection dialog.
TEmbedDrawFMX	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]. EmbedDraw 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\EmbedDraw\FMX\3_2\Source;
- C:\Users\Public\Documents\Mesurasoft\EmbedDraw\FMX\3_2\Source\FMX;
- C:\Users\Public\Documents\Mesurasoft\EmbedDraw\FMX\3_2\Source\FMX\DelphiVersion
- C:\Users\Public\Documents\Mesurasoft\EmbedDraw\FMX\3_2\Source\Shared

B)- Define a variable (for example *$(EMBEDDRAW_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.

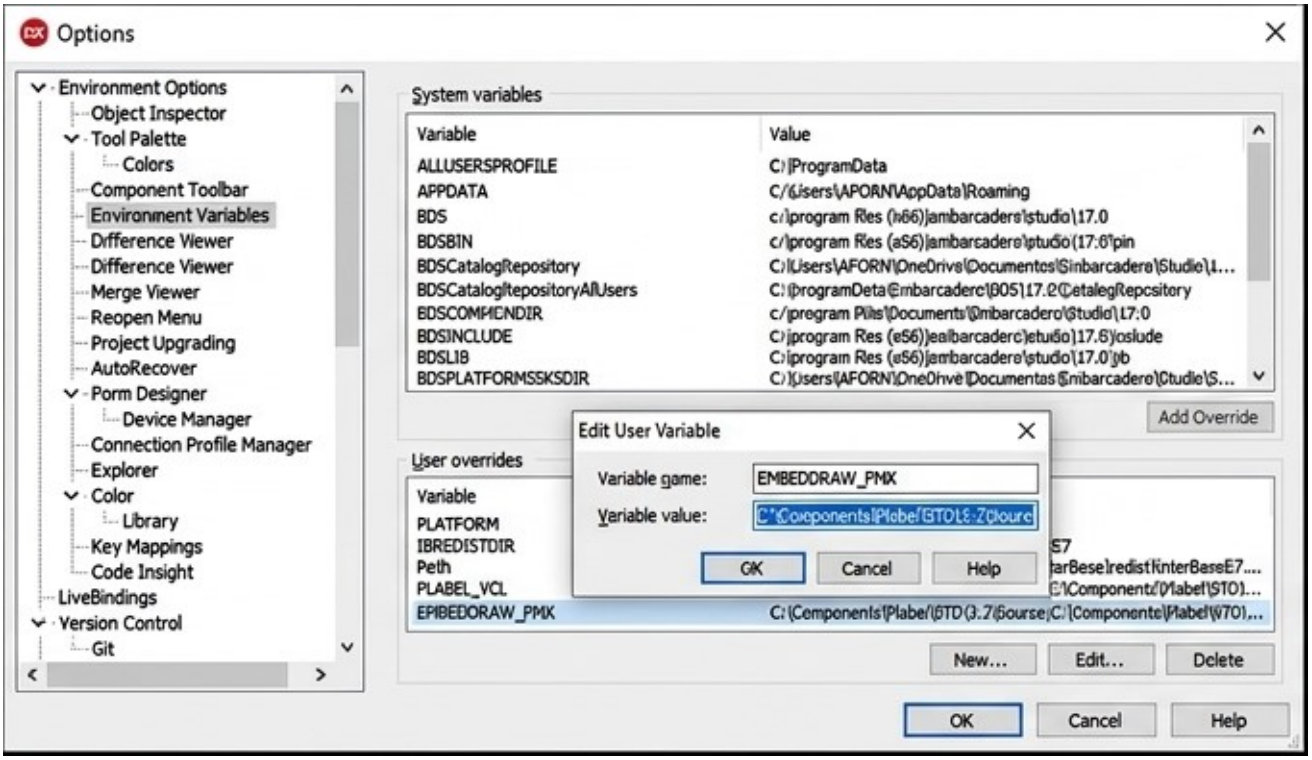


Fig.17 Define FMX variable paths

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

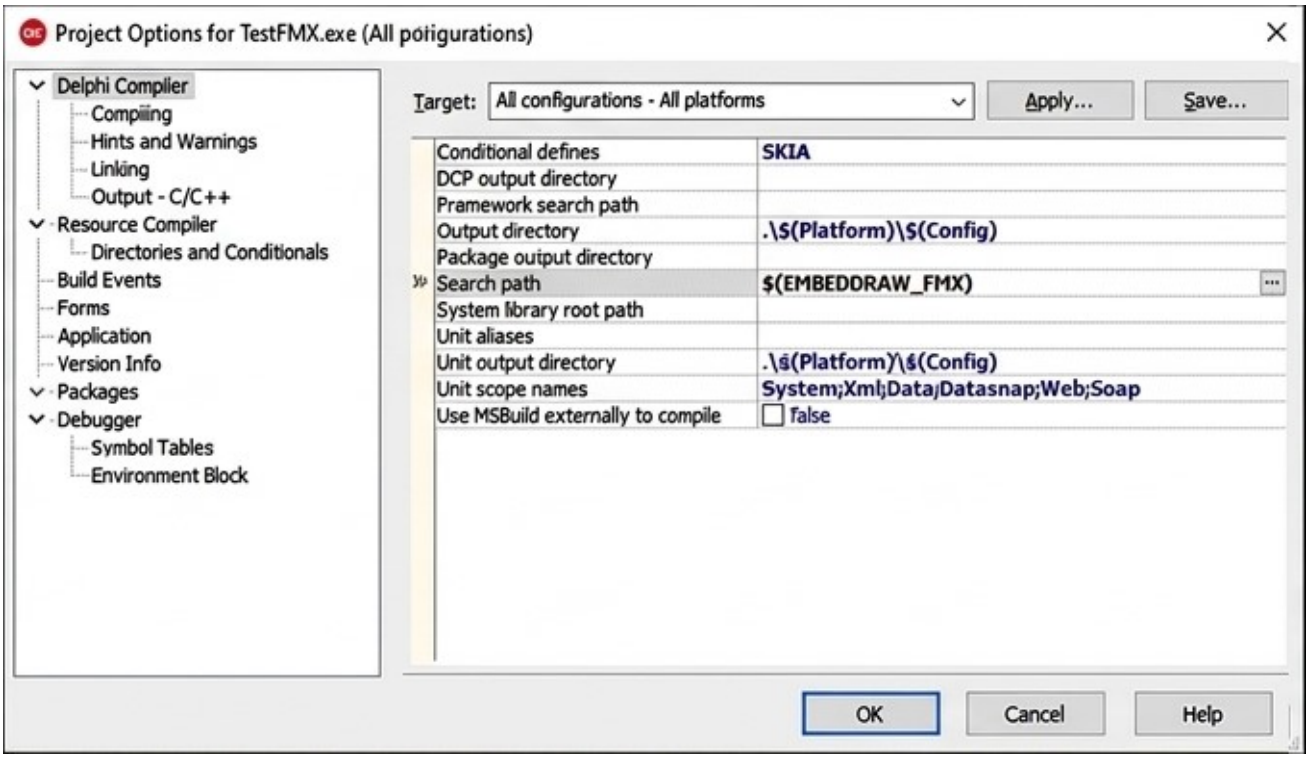


Fig.18 Assign the search path variable

10.5 Compatibility and optional services

- Supported compiler packages are provided from Delphi 10 Seattle (D100) through Delphi 13 Florence (D130), for the applicable Win32/Win64 VCL and FMX targets.
- Skia4Delphi is required for rendering. On Delphi versions where it is not integrated, install the matching Skia4Delphi packages and deploy `sk4d.dll` beside the application executable.
- `TEDWMCPServer` uses the Indy HTTP server units supplied with Delphi. Include the MCP unit and the appropriate VCL or FMX tools unit when using the standard tool set.
- `TMCPAIClient` uses Delphi `System.Net.HttpClient`; outbound HTTPS access and valid provider certificates are required at run time.

11.Internationalization

The **EmbedDraw 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 **TEmbedDrawVCL** and **TEmbedDrawFMX** components. If we do not specify a file, the default language will be English.

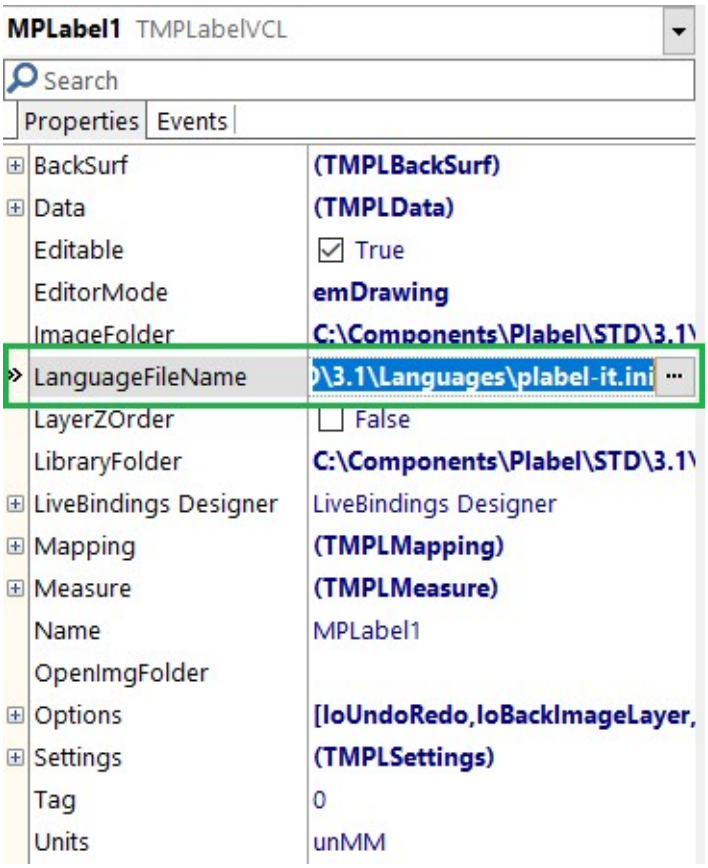


Fig.19 LanguageFileName property

If you cannot find your language in the provided Languages folder's ini files, you can make a copy of EmbedDraw-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.

12.Components

12.1.TEmbedDrawVCL - TEmbedDrawFMX

12.1.1 Properties

12.1.1.1 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.

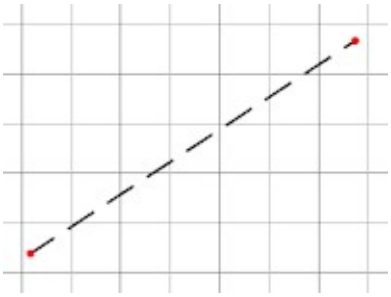
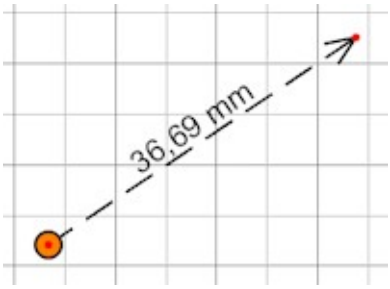
eaLine	eaArrowLine
	

Table3 Line elements I

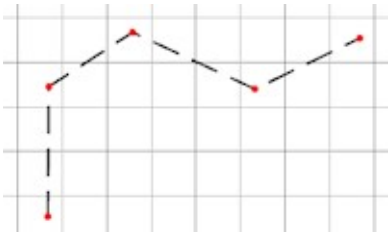
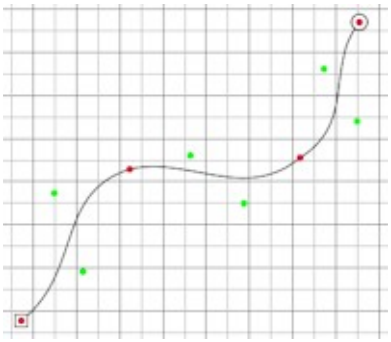
eaPolyline	eaBezier
	

Table4 Line elements II

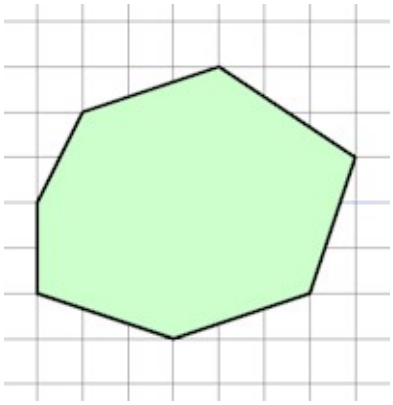
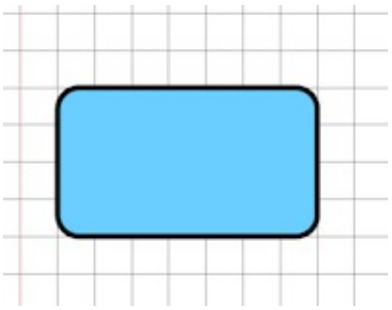
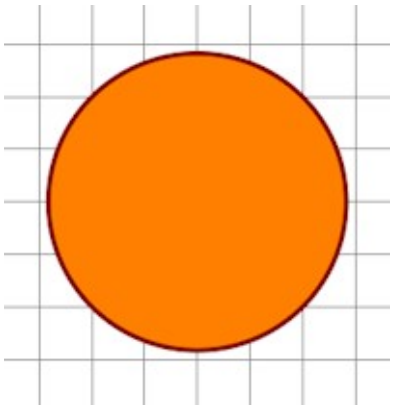
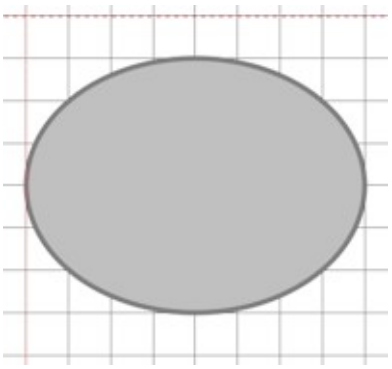
eaPolygon	eaRectangle
	
esCircle (point radio, two points and 3P)	eaEllipse
	

Table5 2D Elements I

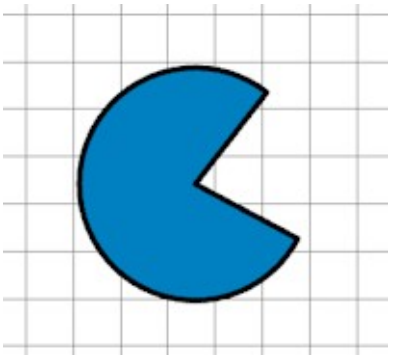
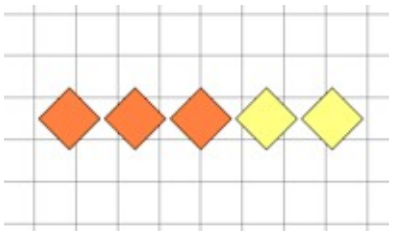
eaSector	eaRank
	

Table6 2D Elements II

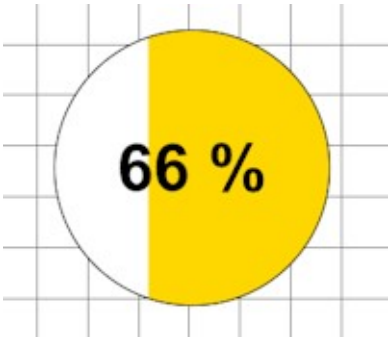
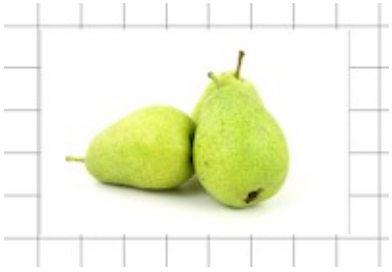
eaLevel	eaImage
	

Table7 2D Elements III

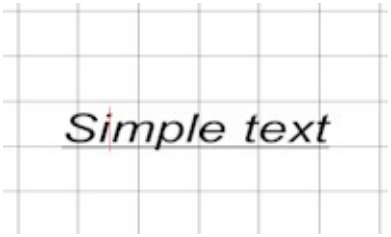
eaText	eaPolytext
	

Table8 Text elements I

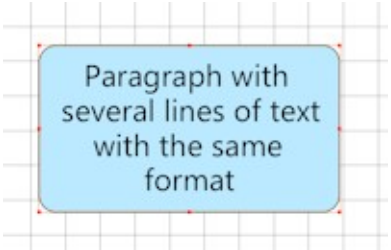
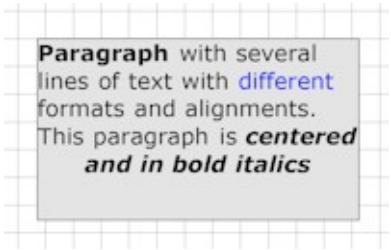
eaParagraph	eaRichText
	

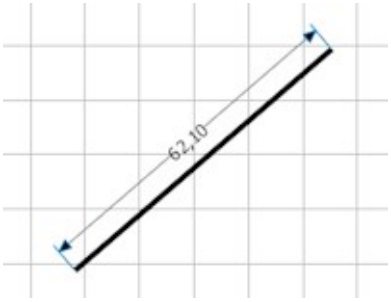
Table9 Text elements II

eaNutriScore	eaSymbol

Table10 Symbolism

eaDim

eaDimAng



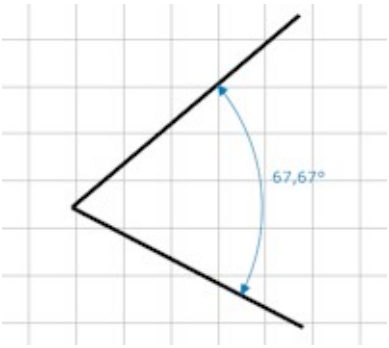
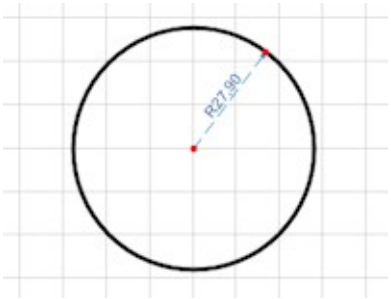


Table11 Dimension types I

eaDimRad1

eaDimRad2



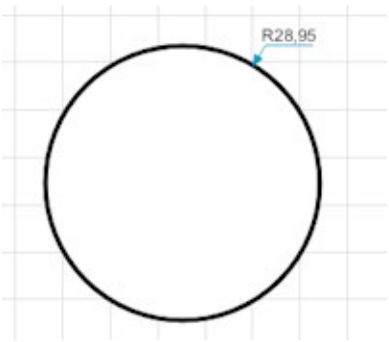
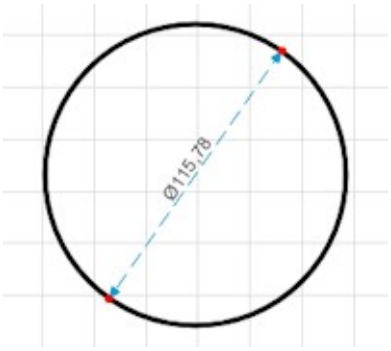


Table12 Dimension types II

eaDimDia1

eaDimDia2



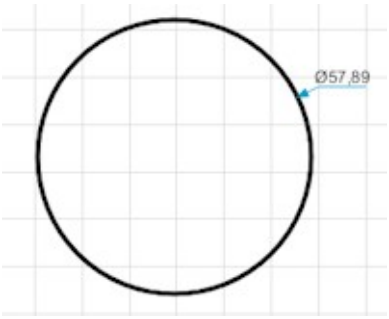


Table13 Dimension types III

eaBarcode (1D)

eaQRCode

Table14 Barcodes I

eaDataMatrix

eaPDF417

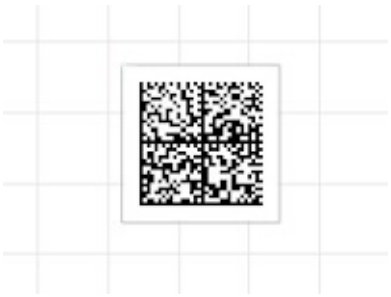
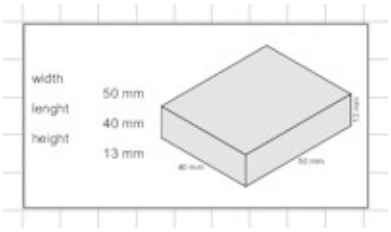


Table15 Barcode II

eaPackage (package 2D or 3D dimension)

eaPackage (2D part with drills)



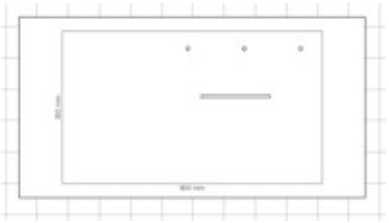


Table16 Package

eaTable

variedad	categoria	cajas	confección
Melocotón	I	40	Caja de cartón 5 Kg Alveolo 46
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.

12.1.1.2 BackSurf.Color

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

12.1.1.3 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 TEDWTable element.

12.1.1.4 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.

12.1.1.5 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).

12.1.1.6 Editable

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

12.1.1.7 ImageFolder

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

12.1.1.8 LanguageFileName

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

12.1.1.9 LayerZOrder

Boolean property in TEmbedDrawVCL and TEmbedDrawFMX 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.

12.1.1.10 LibraryFolder

When we want to insert a TEDWSymbol 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)

12.1.1.11 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 TEDWImage 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).

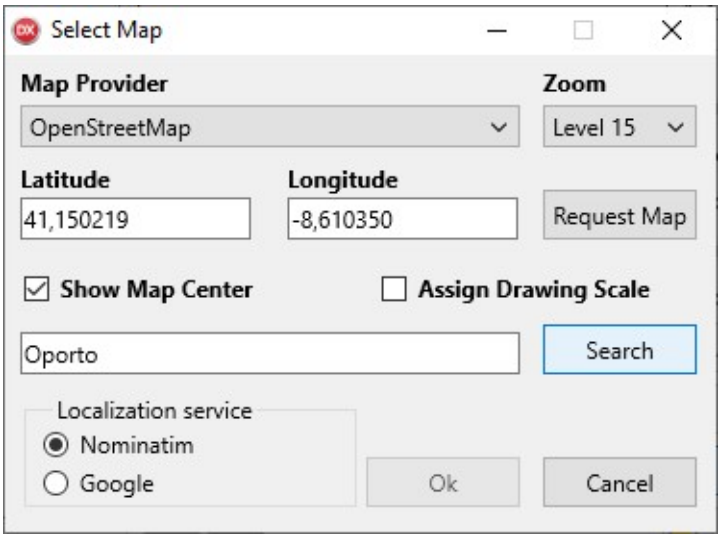


Fig.20 Select Map dialog (Layers editor)

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

12.1.1.12 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);
```

12.1.1.13 OpenImgFolder

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

12.1.1.14 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 (TEDWCmd). 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

12.1.1.15 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

12.1.1.16 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);

12.1.2 Methods

12.1.2.1 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.

12.1.2.2 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 **TEDWFactory** class (*MPL.VCL.Shared.Factory / MPL.FMX.Shared.Factory* unit) to add new elements to the editor, as the previous code.

12.1.2.3 BringToFront

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

12.1.2.4 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.

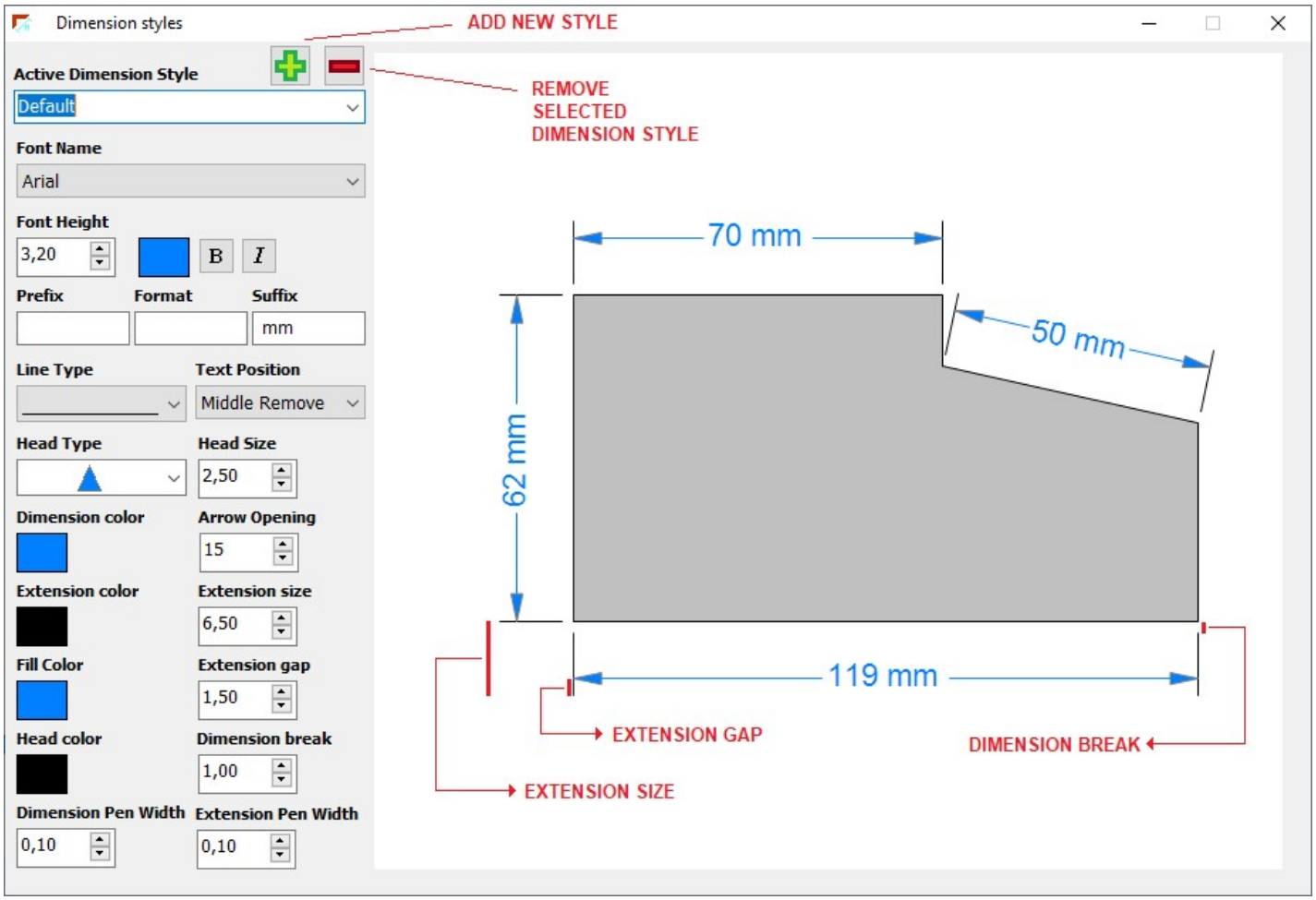


Fig.21 Dimension styles editor

12.1.2.5 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.

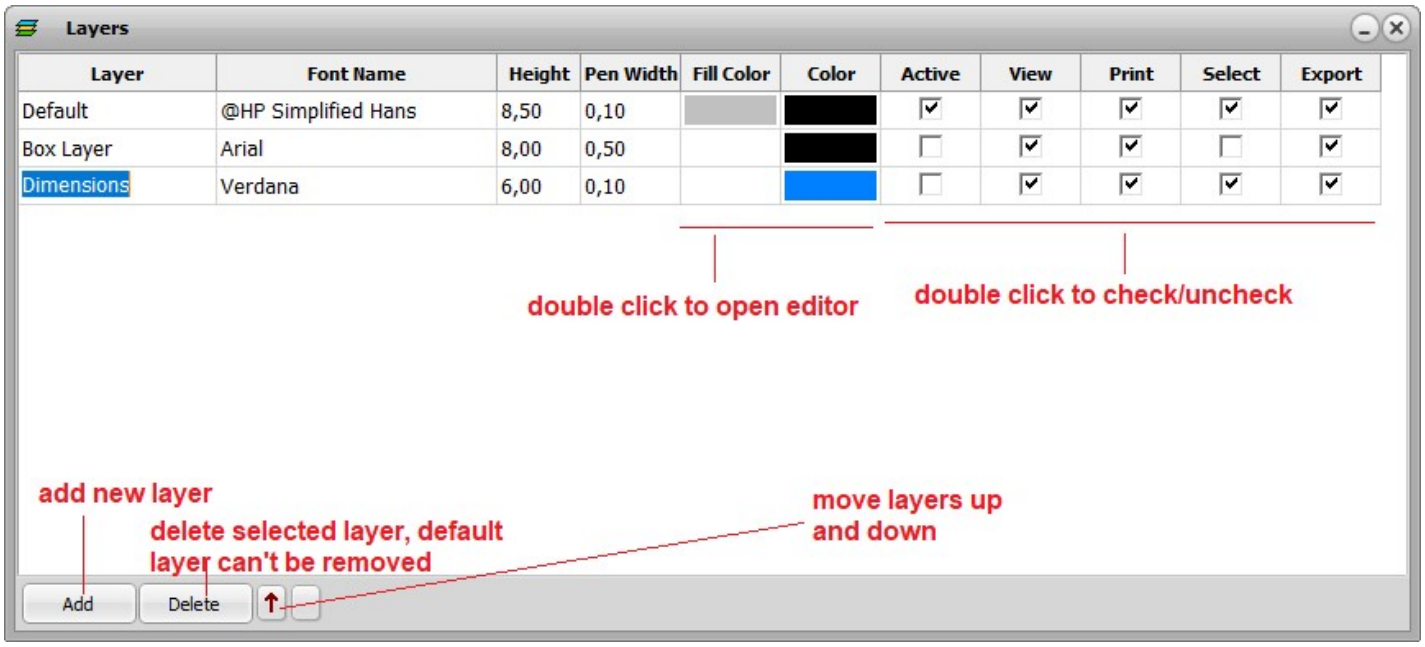


Fig.22 VCL Layer editor

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.

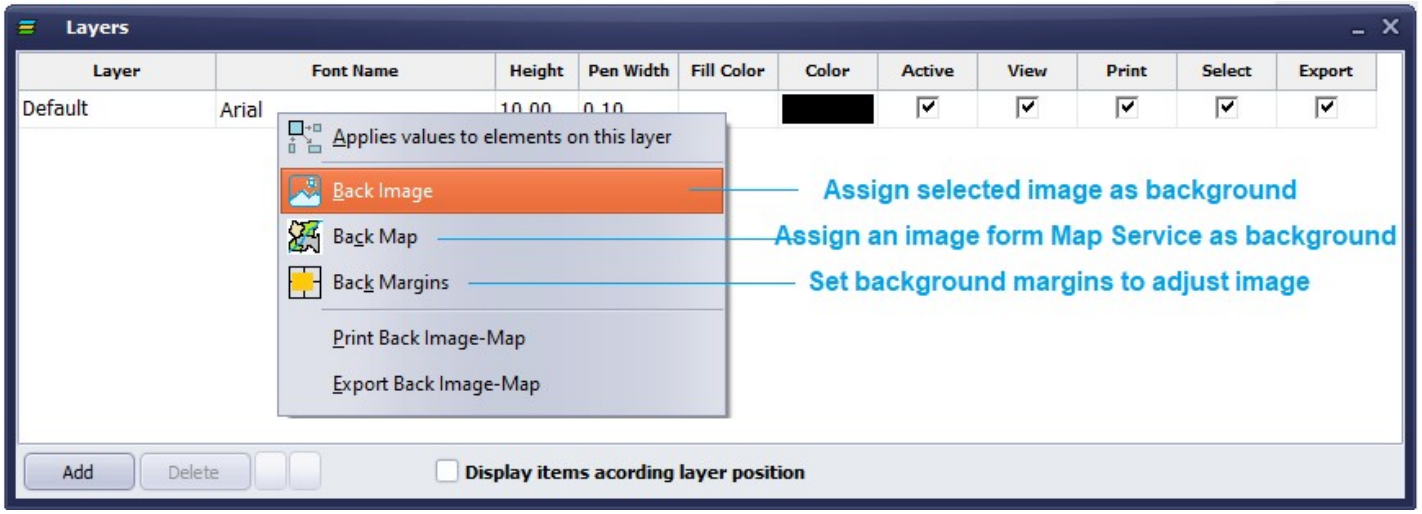


Fig.23 Select background image

12.1.2.6 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*).

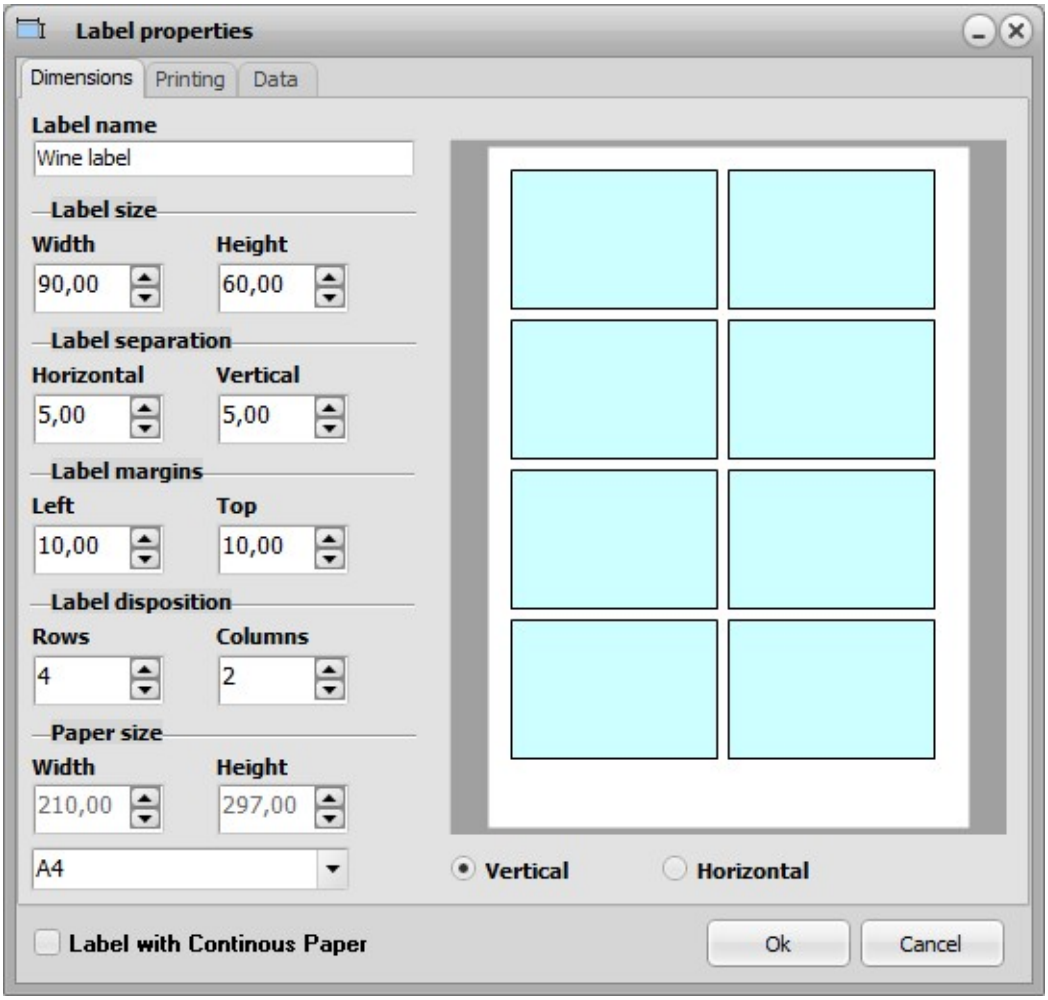


Fig.24 Editor properties for emLabelling mode

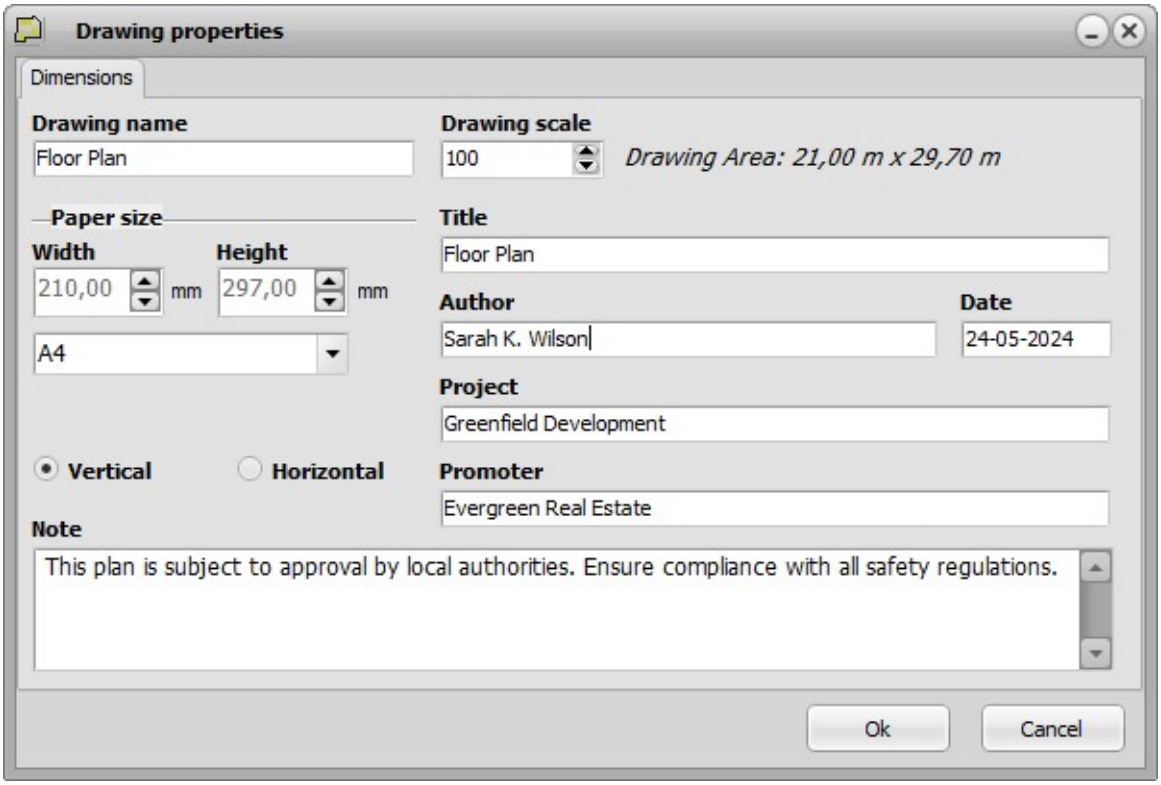


Fig.25 Properties editor for drawing and flow chart mode

12.1.2.7 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.

12.1.2.8 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);
```

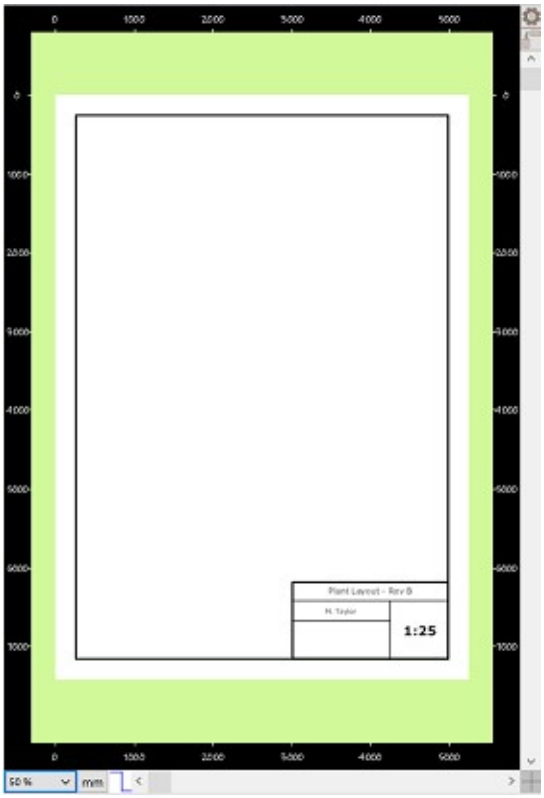


Fig.26 Title block

12.1.2.9 LoadFromFile(const aFilename: TFileName);

We load a previously saved file into the EmbedDraw 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;
```

12.1.2.10 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;
```

12.1.2.11 PasteSelection

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

12.1.2.12 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.

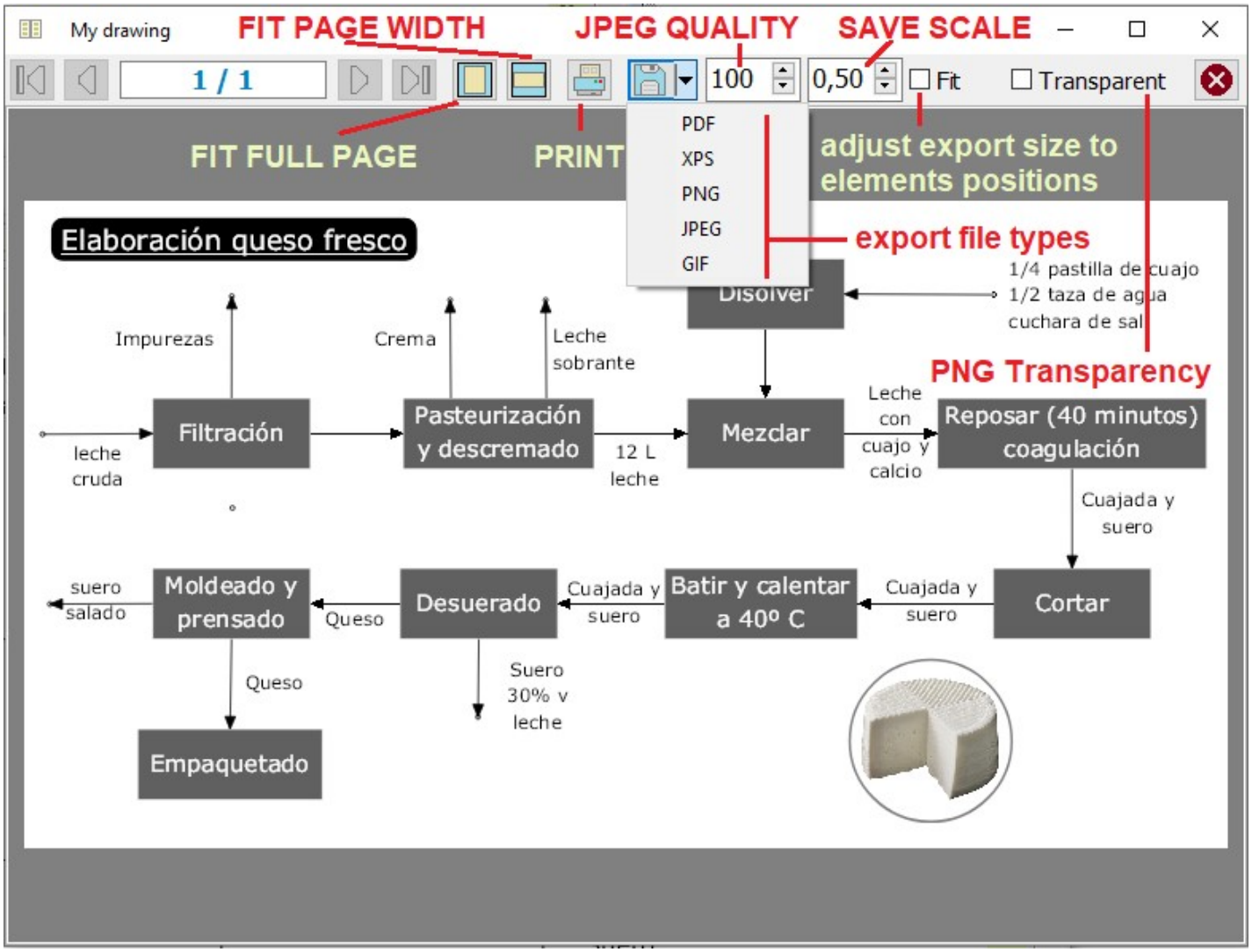


Fig.27 Preview

12.1.2.13 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;
```

12.1.2.14 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.

12.1.2.15 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.

12.1.2.16 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;
```

12.1.2.17 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;
```

12.1.2.18 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;
```

12.1.2.19 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.

12.1.3 Events

12.1.3.1 OnBeforePrint

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

Before starting the printing process (printing thread).

12.1.3.2 OnAfterPrint

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

Upon successful completion of printing.

12.1.3.3 OnGMLCoordinateTransform

Allows TEmbedDrawVCL and TEmbedDrawFMX to import GML files using a coordinate system not supported natively by EmbedDraw.

```
procedure(
  Sender: TObject;
  EPSG: Integer;
  X, Y: Double;
  var Latitude, Longitude: Double;
  var Handled: Boolean
) of object;
```

- EPSG: source coordinate-system code extracted from the GML.

- X, Y: original GML coordinates.

- Latitude, Longitude: resulting WGS84 coordinates in degrees.

- Handled: set to True when the conversion succeeds. Leave it False for unrecognized EPSG codes.

The returned latitude must be between -90 and 90, and longitude between -180 and 180. This example converts a small local metric grid identified as EPSG 99999:

```
uses
  System.Math;

procedure TForm1.GMLCoordinateTransform(
  Sender: TObject;
  EPSG: Integer;
  X, Y: Double;
  var Latitude, Longitude: Double;
  var Handled: Boolean);
const
  OriginX   = 500000.0;
  OriginY   = 4500000.0;
  OriginLat = 40.416775;
  OriginLon = -3.703790;
  MetresPerDegree = 111320.0;
begin
  Handled := False;

  if EPSG <> 99999 then
    Exit;

  Latitude :=
    OriginLat + (Y - OriginY) / MetresPerDegree;

  Longitude :=
    OriginLon + (X - OriginX) /
      (MetresPerDegree * Cos(DegToRad(OriginLat)));

  Handled := True;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  EmbedDrawVCL1.OnGMLCoordinateTransform :=
    GMLCoordinateTransform;

  // For FMX:
  // EmbedDrawFMX1.OnGMLCoordinateTransform :=
  //   GMLCoordinateTransform;
end;
```

For official coordinate systems, use a proper geospatial projection library and return WGS84 latitude/longitude.

12.1.3.4 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.

12.1.3.5 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 **TEmbedDrawVCL or TEmbedDrawFMX** 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;
```

12.1.3.6 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 **TEDWInspector** 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;
```

12.1.3.7 OnUndoRedoEvent

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

```
```

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

12.1.3.8 OnGetListValues

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

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

12.1.3.9 OnSelectItem

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

aItem : selected item.

When an object enters selection.

12.1.3.10 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.

12.6.TEmbedDraw public API

TEmbedDrawVCL and **TEmbedDrawFMX** expose the same document API through **TEDWBaseLabel**. The following public members complement the component properties already described in this manual.

12.3 Document and serialization methods

Member	Result	Description
`GetJson: TJSONObject`	Owned JSON object	Builds the complete JSON representation of the current drawing. The caller must release the returned object.
`GetJsonString(Compact: Boolean): string`	JSON text	Returns the current drawing as text. Set `Compact` to False for readable formatted output.
`SaveSVG(FileName, ExportType, Factor, AttrID)`	None	Writes vector svg output. `Factor` controls scale and `AttrID` emits element identifiers for later DOM processing.
`SelectionBounds(AType): Int64`	Base coordinate	Returns a requested boundary of the current selection in internal units (0.001 mm).
`CopySelection` / `PasteSelection`	None	Copies and restores the selected elements using the platform clipboard implementation.
`BeginUpdate(PaintOnEnd)` / `EndUpdate`	None	Groups multiple changes and suppresses intermediate repaints. Always pair both calls with `try/finally`.

12.4 Selection and arrangement

Member	Description
`AlignObjects(AType)`	Aligns selected objects according to `TMPLObjectAlign`.
`SpaceHorizontal(AGap = -1)` / `SpaceVertical(AGap = -1)`	Distributes selected objects. A value of -1 calculates equal spacing between the first and last object.
`CenterHorizontal` / `CenterVertical`	Centers the selected objects relative to their combined bounds.
`SameWidth` / `SameHeight`	Applies a common dimension to all selected objects.
`ElementByID(ID, Select = True)`	Finds an element by its string identifier and optionally selects it.
`DeleteSelection`	Removes all selected elements and records the operation in the undo history.

12.5 Layer methods

Member	Description
`AddLayer: TEDWLayer`	Creates a layer and returns it for immediate configuration.
`LayerIdx(Layer): SmallInt`	Returns the zero-based internal layer index.
`SetActiveLayer(Index): Boolean`	Activates a valid layer and reports success.
`DeleteLayer(Index): Boolean`	Deletes a layer when the operation is valid and reports success.
`ExchangeLayers(Index1, Index2): Boolean`	Changes layer order.
`GetLayersNames(List)`	Fills an existing `TStrings` instance with layer names.

12.6 Read-only state and indexed collections

Property	Type	Description
`Modified`	Boolean	Indicates that the drawing has changed since its last reset or save operation.
`ItemsCount` / `Item[Index]`	Integer / TEDWBaseItem	Enumerates all drawing elements.
`SelectionCount` / `Selection[Index]`	Integer / TEDWBaseItem	Enumerates selected elements.
`LayersCount` / `Layer[Index]`	Integer / TEDWLayer	Enumerates layers.
`ActiveLayer`	TEDWLayer	Returns the current destination layer.
`EditorState` / `States`	State types	Expose current editor activity and state flags for command enabling.

```
EmbedDraw1.BeginUpdate(False);
try
  L := EmbedDraw1.AddLayer;
  L.Name := 'Annotations';
  EmbedDraw1.SetActiveLayer(EmbedDraw1.LayerIdx(L));
  // Add or modify several elements here.
finally
  EmbedDraw1.EndUpdate;
end;
```

12.10.Document configuration objects

The main drawing exposes persistent configuration objects through `Measure`, `BackSurf`, `Settings`, `Mapping` and indexed `Layer` properties. Assign their values rather than replacing their owned instances.

12.8 TEDWMeasure

Group	Members	Description
Paper and labels	`PaperName`, `PaperSize`, `PaperWidth`, `PaperHeight`, `Rows`, `Columns`, `Width`, `Height`	Define sheet and repeated-label geometry.
Margins and spacing	`MarginLeft`, `MarginTop`, `GapRows`, `GapColumns`, `InnerCircle`	Control placement inside the paper.
Drawing scale	`DrawScale`, `GetSize`, `WidthUnits`, `HeightUnits`	Convert stored dimensions and expose the scaled document size.
Convenience methods	`SetPaperSizeMM`, `SetLabelSizeMM`, `Reset`	Configure sizes in millimetres or restore measurement defaults.
Metadata	`Promoter`, `Author`, `Project`, `Title`, `Date`, `Note`, `PlanNumber`	Store plan/document identification.

`ContinousPaper` retains this spelling for source compatibility.

12.9 TEDWLayer

Group	Members	Description
Identity and order	`Name`, `ZPos`, `Options`	Name, stacking order and selectable/print/export/background behavior.
Drawing style	`Color`, `BackColor`, `Fill`, `PenWidth`, `LineType`, font properties	Default appearance inherited by suitable items.
Image content	`FileImg`, `Embeded`, `PrintImg`, `ImageBytes`, image margins	External or embedded layer image configuration.
Map persistence	`MapProvider`, `MapZoom`, `MapLatitude`, `MapLongitude`, `ShowMapCenter`	Stores the exact provider and map request used by GML/GPX and manual map layers.

12.10 TEDWMapping and related settings

`TEDWMapping` exposes `Latitude`, `Longitude`, `Zoom`, `Provider`, `MapProviders`, `MapApi`, `ShowCenter`, `Loading` and the generated `FileName`. `LoadMap` starts map retrieval, while `Tiles` and `NumTiles` expose the current tile set. `TEDWBackSurf` controls background style, image, alignment and printing; `TEDWSettings` controls link gaps and terminal size/colors.

12.15.TEDWData public API

TEDWData, available through `TEmbedDrawVCL/FMX.Data`, manages JSON, CSV, Delphi datasets, detail records, prompts and reusable value lists. It is owned by the drawing component.

12.12 Data-source properties

Property	Description
`DataType`	Indicates the active source kind.
`Json` / `SaveJson`	JSON source text and whether it is persisted with the drawing.
`Datasets`	Collection of Delphi dataset links and their detail definitions.
`CsvFileName`, `CsvSep`, `CsvHeader`, `CsvQuote`	CSV location and parsing rules.
`LinkData`	Enables evaluation of data-linked drawing fields.
`FieldRepeat`	Name of an integer field that controls record repetition.
`DataSel`	Selects the current dataset/data source entry.

12.13 Reading values

Member	Description
`DataValue(Name, Format)`	Returns a formatted value from the current source.
`DataDetailValue(Detail, Name, Format)`	Returns a field from the current detail record.
`DataList`, `DataAsk`, `DataAskInt`, `DataAskNum`	Read reusable list values and runtime prompt answers.
`DataGraphic(Name, Stream)`	Writes binary graphic data into the supplied memory stream and reports success.
`GetDataset`, `CurrentDataset`, `GetDetail`	Access configured dataset and detail objects.
`GetDetailCalc`	Returns a formatted detail aggregate according to `TMPLFooter`.

12.14 CSV and runtime configuration

Member	Description
`SetCsvData(File, HasHeader, Separator, Quote)`	Loads/configures CSV data and reports whether it is usable.
`SetCsvField(Position, FieldType)`	Assigns a type to a CSV column.
`AddListVal` / `DeleteList`	Add or remove reusable value-list definitions.

`AddAskVal` / `DeleteAsk`	Add or remove runtime question definitions.
`ClearListVal` / `ClearAskList`	Clear those collections.
`IsValidData(FillType, Name)`	Checks whether a named field/value is valid for a fill operation.
`FirstDetail` / `NextDetail`	Navigate a named detail dataset.

12.15 State properties

`HasData`, `HasList`, `HasQuest`, `Numlabels`, `NumListVal` and `NumQuestions` describe current state. Indexed `ListVal` and `Questions` expose definitions. `Pos` is write-only and changes the active record position; validate the source and range before assigning it.

12.16.TEDWInspector

The **TEDWInspector** component is used in conjunction with the **TEmbedDrawVCL** or **TEmbedDrawFMX** components (using EmbedDraw 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.

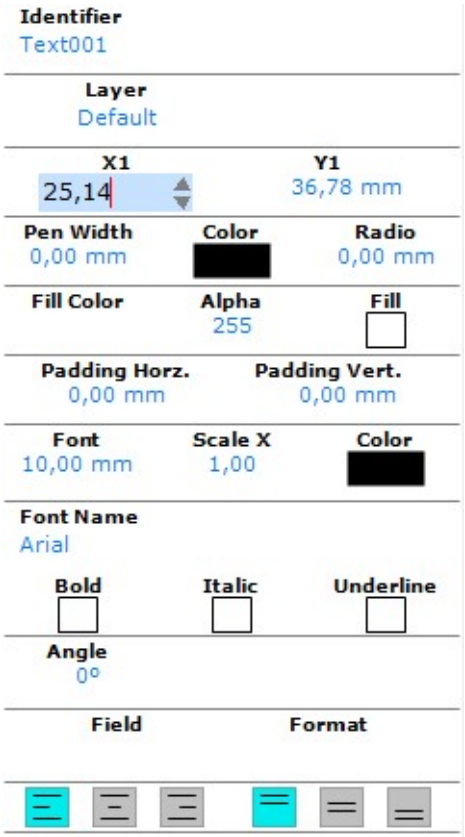


Fig.28 Inspector component

12.16.1 Properties

12.16.1.1 EmbedDraw

Links the inspector to a **TEDWBaseLabel** (or descendant such as **TEmbedDrawVCL** / **TEmbedDrawFMX**) 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.

12.16.1.2 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

12.16.1.3 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

12.16.1.4 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.

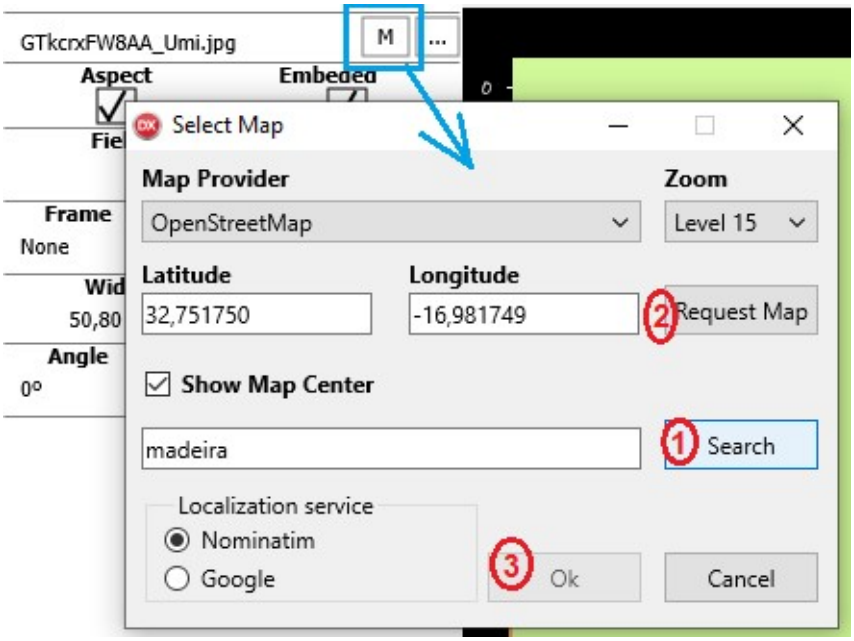


Fig.29 Select map image button

12.16.1.5 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

12.16.1.6 TitleColor

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

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

12.16.1.7 ValueColor

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

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

12.16.1.8 EditColor

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

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

12.16.1.9 SelColor

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

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

12.16.1.10 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

12.16.1.11 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

12.17.TEDWEditor

TEDWEditor is the visual label/drawing/flowchart editor panel. It provides scrollbars, rulers, zoom, grid, snap and inline editing around a linked **TEmbedDrawVCL** / **TEmbedDrawFMX** 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.

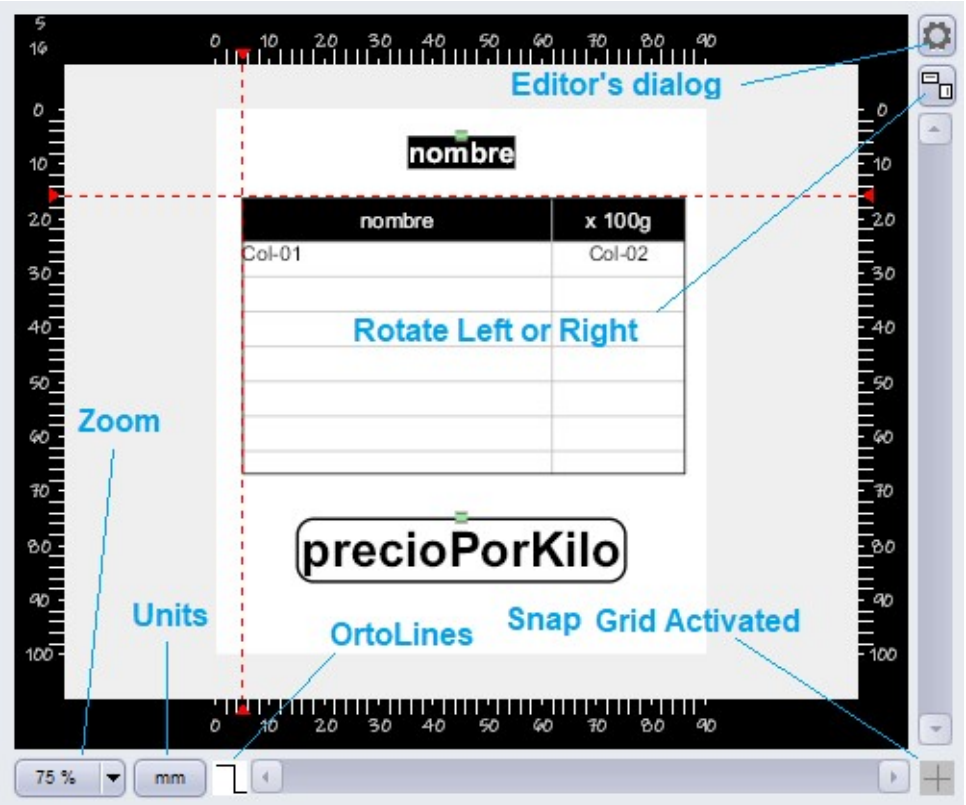


Fig.30 Editor

12.17.1 Properties

12.17.1.1 EmbedDraw

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: TEmbedDrawVCL
- FMX: TEmbedDrawFMX

12.17.1.2 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.

12.17.1.3 LabelPosition

Controls where the label or drawing page is placed inside the editor viewport. `lbTopLeft` is the default and keeps the page at the upper-left margin; `lbCenter` centres it horizontally and vertically whenever the viewport is larger than the scaled page. This property changes only the editor presentation and does not alter coordinates,

printing or exported output.

12.17.1.4 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

12.17.1.5 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

12.17.1.6 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

12.17.1.7 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.

12.17.1.8 MoveY

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

12.17.1.9 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

12.17.1.10 OrtoLines

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

12.17.1.11 Margin

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

12.17.1.12 MarginColor

Fill colour of the margin area surrounding the label.

12.17.1.13 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.

12.17.1.14 SizeRef

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

12.17.1.15 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.31 Align label topleft

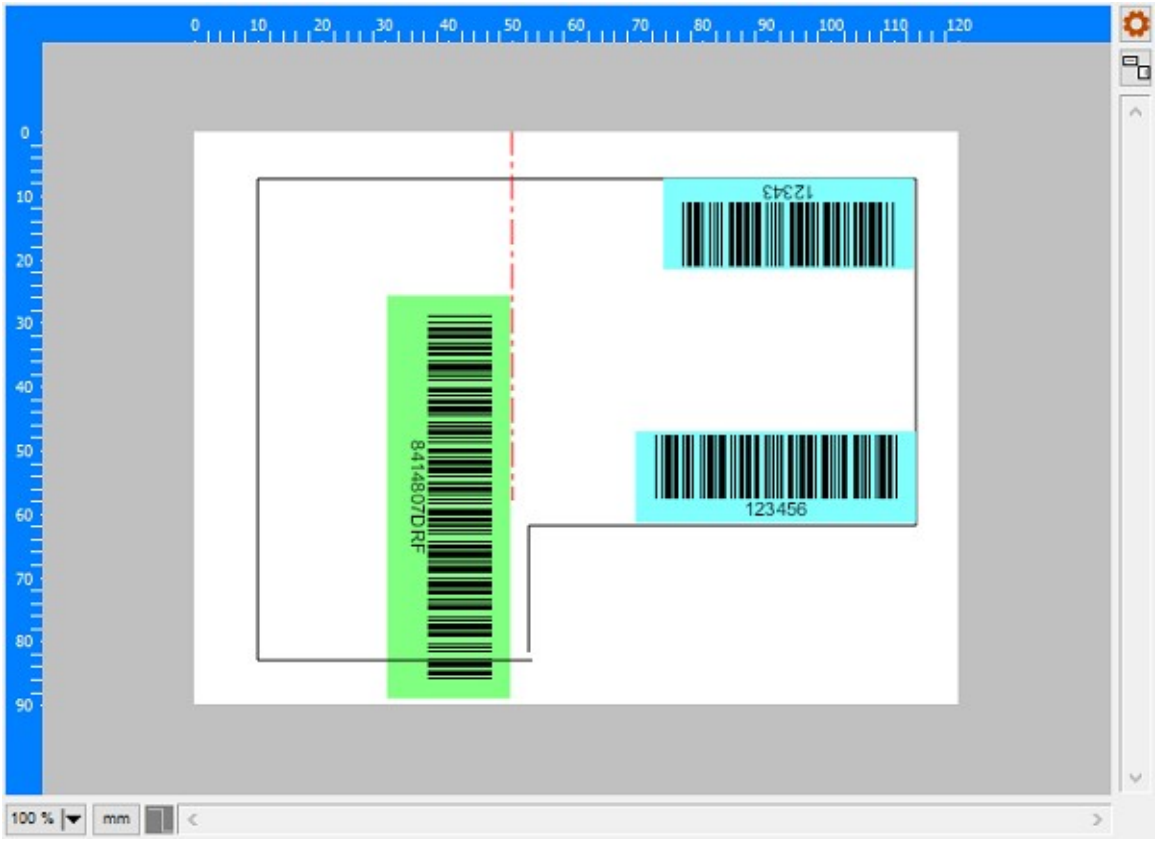


Fig.32 Align label centered

12.17.1.16 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 TEDWRichText 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

12.17.1.17 EnabledMouseFilter

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

12.17.1.18 ShowRefLines

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

12.17.2 Methods

12.17.2.1 BeginUpdate

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

12.17.2.2 EndUpdate

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

12.17.2.3 EditSettings

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

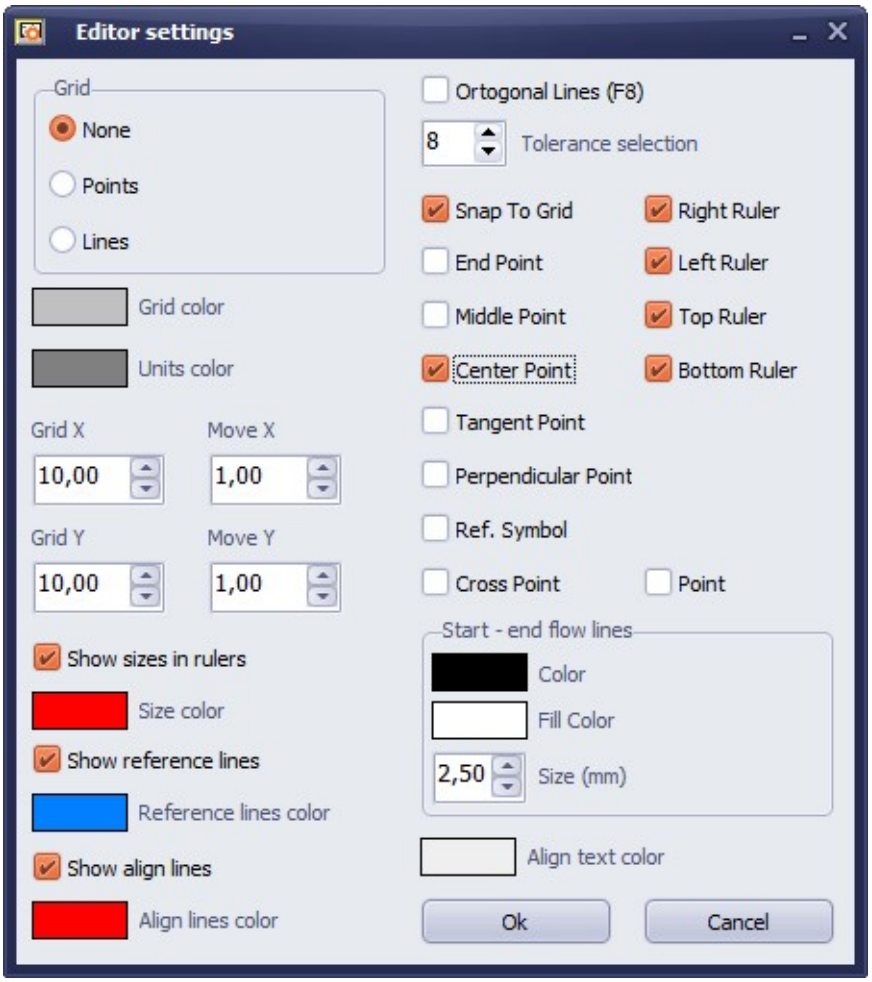


Fig.33 Editor's settings dialog

12.17.2.4 AlignSelectedLeft

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

12.17.2.5 AlignSelectedRight

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

12.17.2.6 AlignSelectedTop

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

12.17.2.7 AlignSelectedBottom

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

activated).

12.17.2.8 ScrollToItem

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

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
 EDWEditor1.ScrollToItem(EmbedDraw1.ElementByID('test'));
end;
```

12.17.3 Events

12.17.3.1 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).

12.17.3.2 OnMouseUp

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

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

12.17.3.3 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.

12.18.TEDWPreview

TEDWPreview renders a live overview of a **TEmbedDrawVCL** or **TEmbedDrawFMX** document. Assign `EmbedDraw` to connect it; notifications keep the preview synchronized with document changes.

Member	Type/default	Description
`EmbedDraw`	TEmbedDrawVCL / TEmbedDrawFMX	Drawing displayed by the preview.
`ViewType`	TMPLViewType; `vtFitPage`	Selects how the drawing is fitted in the available control area.
`ContentWidth`	Integer; 0	Explicit content width. Zero lets the control derive the value automatically.
`ContentHeight`	Integer; 0	Explicit content height. Zero enables automatic calculation.
`BackColor`	TColor /TAlphaColor	Background outside the rendered drawing.
`Margin`	Integer; 10	Space around previewed content.
`BorderStyle`	TBorderStyle; `bsSingle`	VCL-only border style.
`Draw`	Method (VCL)	Forces immediate preview rendering when an explicit refresh is required.

```
EDWPreview1.EmbedDraw := EmbedDraw1;
EDWPreview1.ViewType := vtFitPage;
EDWPreview1.Margin := 12;
```

12.21.TEDWViewLibrary

TEDWViewLibrary displays symbols stored in an EmbedDraw library and lets the user select one for insertion. VCL and FMX expose the same library workflow.

12.20 Methods

Member	Description
`GetCatList(const AFile: TFileName): string`	Reads the available categories from a library file and returns them as library-defined text.
`LoadSymbols(const AFile: TFileName; const ACategory: string)`	Loads and displays symbols belonging to the requested category.

12.21 Properties and event

Property	Type/default	Description
`EmbedDraw`	TEDWBaseLabel	Destination drawing for the chosen symbol.
`Gap`	Integer; 10	Spacing between symbol cells.
`SymbolWidth` / `SymbolHeight`	Integer; 80	Size of each symbol preview cell.
`IdSelection`	Integer, read-only	Index of the selected symbol.
`Selection`	TEDWBaseLibSymbol, read-only	Selected library symbol, or nil if there is no valid selection.
`ShowHintOnMove`	Boolean; True	Shows symbol information while the pointer moves across cells.
`FrameColor` / `SelectionColor`	Platform color	Frame and current-selection colors. SelectionColor is available through the base class.
`OnChangePos`	TMPLInfoSymEvent	Fires whenever the selected library position changes.

```
EDWViewLibrary1.EmbedDraw := EmbedDraw1;
EDWViewLibrary1.LoadSymbols('symbols.edl', 'Electrical');
if EDWViewLibrary1.Selection <> nil then
  EmbedDraw1.CurrentLibSym := EDWViewLibrary1.Selection;
```

12.22.TEDWSpinEditFloat

TEDWSpinEditFloat is the VCL numeric editor used by EmbedDraw dialogs and can also be placed on application forms. It extends `TCustomEdit` with floating-point validation and spin buttons.

Property	Type	Description
`Value`	Double	Current floating-point value.
`AsInteger`	Integer	Integer representation for code that requires whole values.
`Increment`	Double	Amount added or subtracted by each spin action.
`MinValue` / `MaxValue`	Double	Accepted value range.
`Digits`	Integer	Number of visible decimal digits.
`Precision`	Integer	Formatting and parsing precision.
`EditorEnabled`	Boolean	Allows or prevents direct keyboard editing while retaining spin behavior.
`Button`	TWinControl, read-only	Provides access to the associated spin-button control.

```
EDWSpinEditFloat1.MinValue := 0;
EDWSpinEditFloat1.MaxValue := 100;
EDWSpinEditFloat1.Increment := 0.25;
EDWSpinEditFloat1.Digits := 2;
```

12.23.TEDWFactory

TEDWFactory , declared separately for the VCL and FMX shared units, creates drawing elements with consistent initialization. Methods whose coordinates are `Single` use the document/display units; their `AddBase...` counterparts use internal `Int64` coordinates in 0.001 mm.

Group	Public methods	Purpose
Text	`AddText` , `AddParagraph` , `AddRichText` , `AddPolyText`	Create text elements with the appropriate geometry.
Shapes	`AddPoint` , `AddLine` , `AddArrowLine` , `AddPolyLine` , `AddBezier` , `AddArc` , `AddRectangle` , `AddPolygon` , `AddCircle` , `AddSector` , `AddEllipse`	Create geometric primitives.
Structured elements	`AddTable` , `AddPackage` , `AddRank` , `AddNutriScore` , `AddBarcode` , `AddPDF417` , `AddDataMatrix` , `AddQRCode`	Create application and coding elements.
Symbols and flowcharts	`AddSymbol` , `AddFlow` , `AddLink` , `AddPolyLink`	Create symbols, flowchart nodes and connectors.
Dimensions	`AddDim` , `AddDimAngle` , `AddDimRadial1/2` , `AddDimDiameter1/2`	Create linear, angular, radial and diameter dimensions.
Diagram import	`LoadDiagram` , `LoadMermaid` , `LoadGantt`	Replace document content from supported diagram text. `LoadMermaid` is the compatibility wrapper; use `LoadDiagram` for Mermaid or PlantUML.
DPI	`SetDPI` , `DPIX` , `DPIY`	Set and inspect the factory conversion resolution.

```
Item := TEDWFactory.AddRichText(EmbedDraw1, 10, 10, 80, 30);
Item.ID := 'description';
TEDWFactory.AddQRCode(EmbedDraw1, 100, 10, 30, 30);
```

13.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.

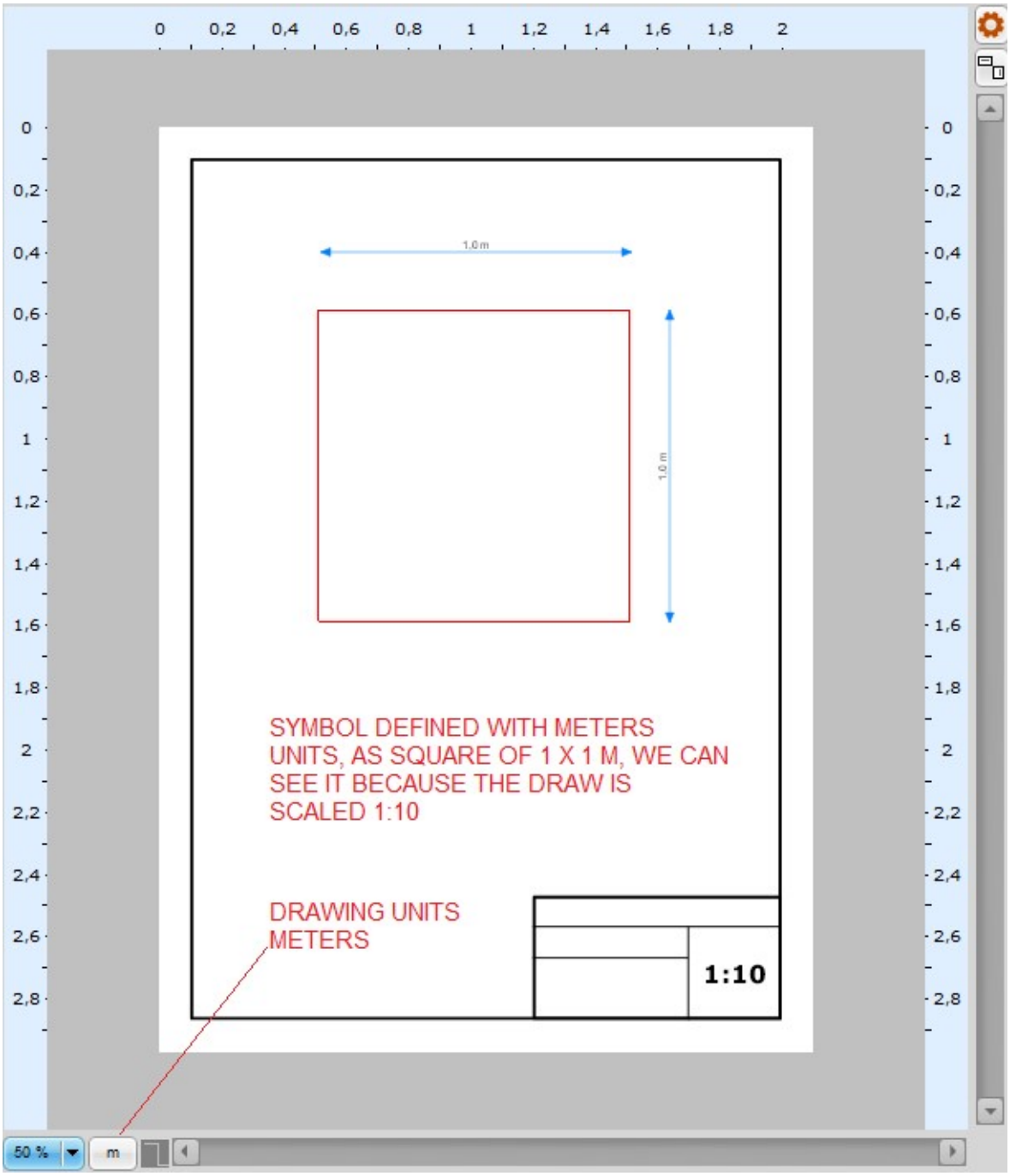


Fig.34 Symbol defined as square with 1 x 1 m

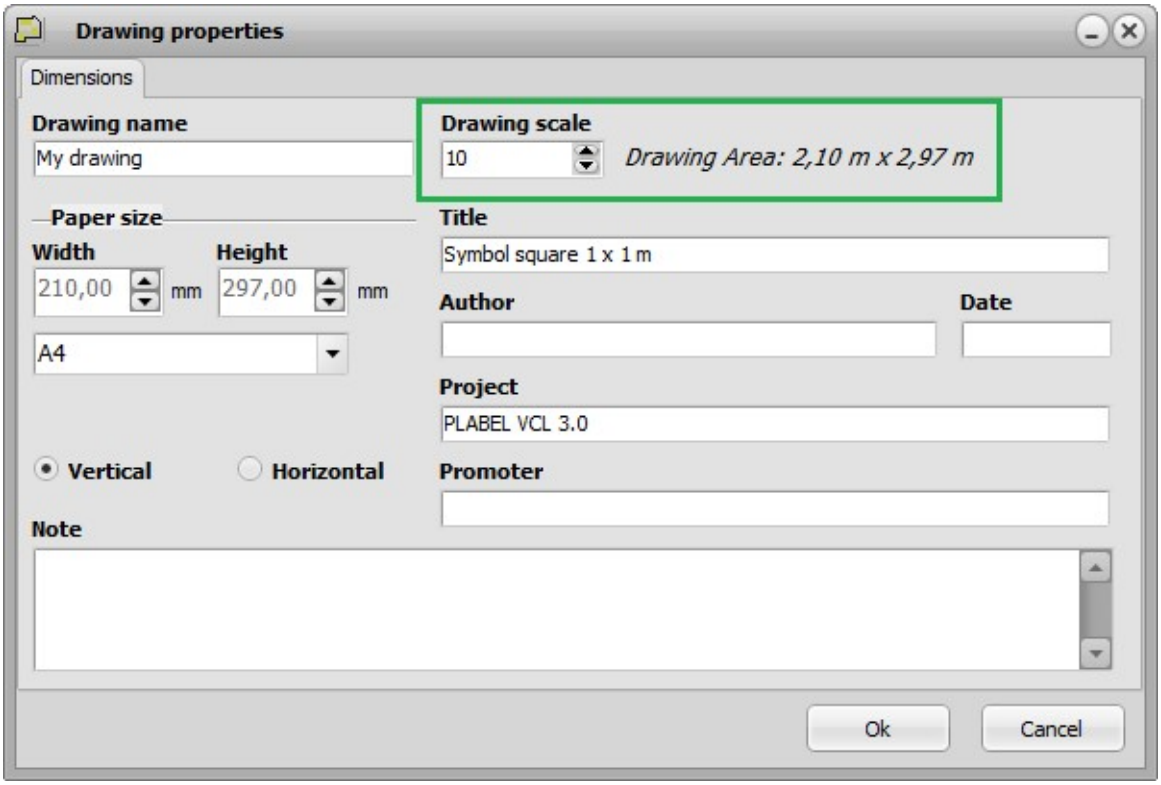


Fig.35 Drawing properties, setting 1:10 scale

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.

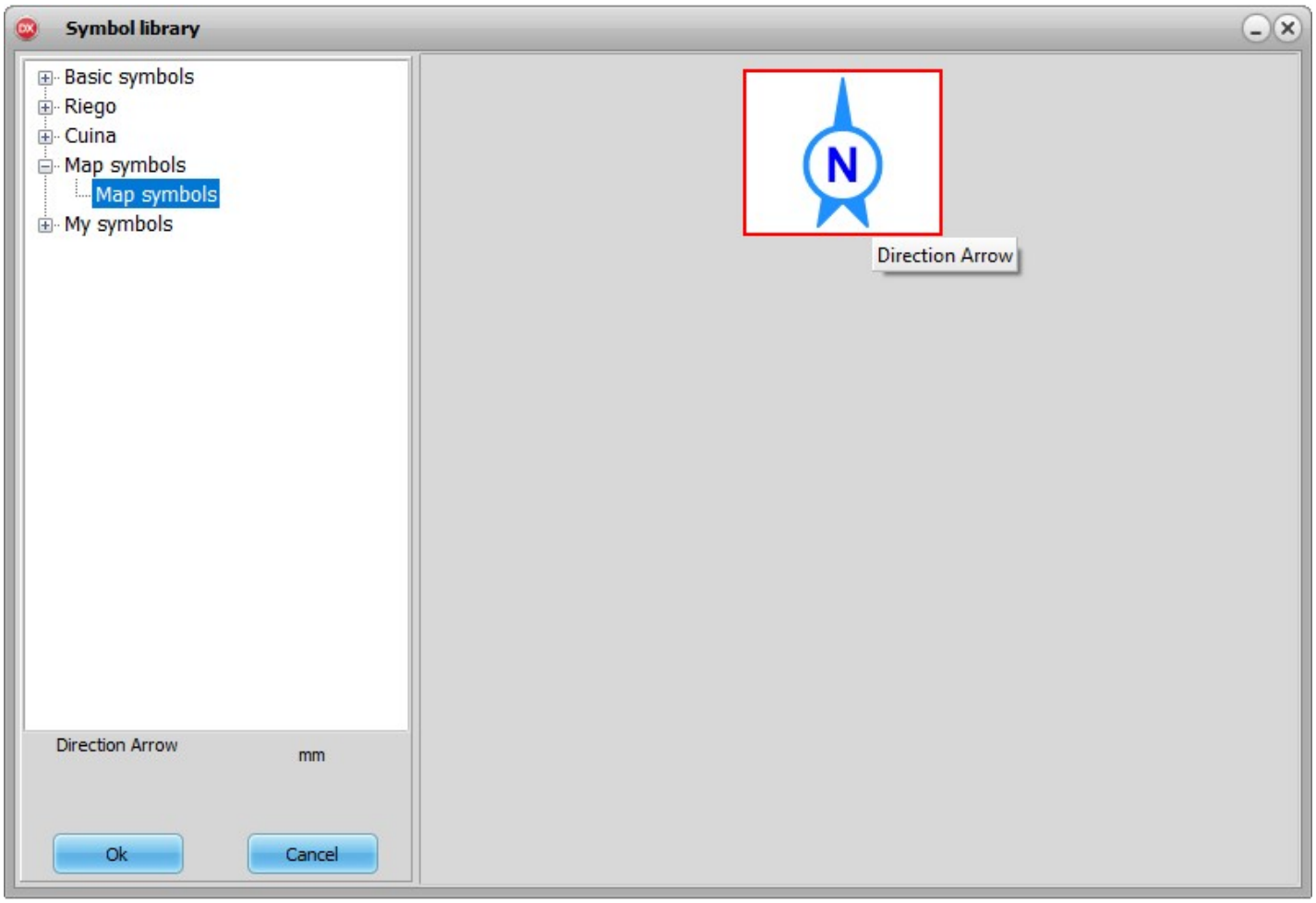


Fig.36 Map direction arrow

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('Map symbols','Direction Arrow','',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 EmbedDraw component are read.

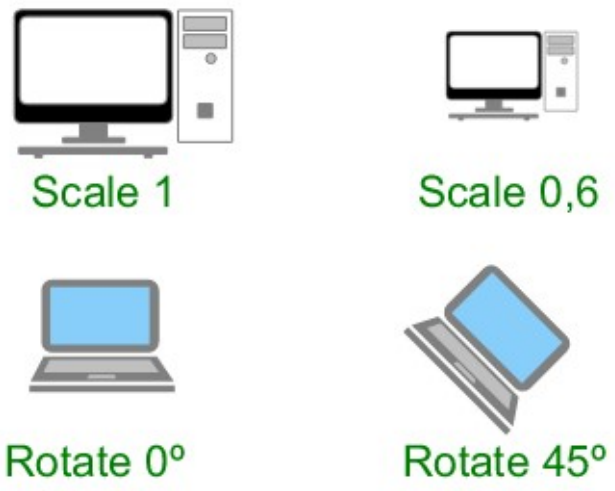


Fig.37 Scale and rotate symbol

13.1 Available symbol primitives

`TEDWLibSymbol` uses a fluent API: every method appends one instruction and returns the same symbol, so calls can be chained. Instructions are executed in insertion order. The following drawing primitives can be used to define a symbol:

Method	Geometry and parameters	Notes
`Line(X1, Y1, X2, Y2)`	Straight segment between two points.	Can also be included in an open compound path.
`Circle(CX, CY, R)`	Circle defined by centre and radius.	Uses the current fill, foreground and background settings.
`Sector(CX, CY, R, Start, Sweep)`	Circular sector; angles are expressed in degrees.	May be filled and stroked.
`Ellipse(CX, CY, RX, RY)`	Ellipse defined by centre and horizontal/vertical radii.	May be filled and stroked.
`Rectangle(X, Y, Width, Height)`	Rectangle defined by origin and size.	This four-parameter overload has square corners.
`Rectangle(X1, Y1, X2, Y2, Radius)`	Rectangle defined by opposite corners.	`Radius = 0` produces square corners; a positive value rounds them.
`Polyline([X1,Y1, ...])`	Open sequence of points.	The array must contain coordinate pairs.
`Polygon([X1,Y1, ...])`	Closed polygon.	The array must contain coordinate pairs; filling uses the current background colour.
`Bezier([X1,Y1, ...], Close)`	One or more cubic Bezier segments.	After the initial point, supply groups of two control points and one end point. `Close` closes the figure and allows filling.
`Arc(X1, Y1, X2, Y2, X3, Y3)`	Circular arc calculated from three points.	Can also be included in an open compound path.
`Text(X, Y, Text, HorzAlign, VertAlign, Angle)`	Text anchored at a point, with horizontal /vertical alignment.	`Angle` is optional and expressed in degrees.

There are no symbol-definition methods for images, barcodes, paragraphs or the higher-level drawing objects. A reusable symbol is composed from the geometric and text primitives listed above.

13.2 Style and construction instructions

Methods	Purpose

`LineWidth`, `LineType`, `ForeColor`, `BackColor`, `Filled`, `RoundCap`, `ScalePen`	Change the current stroke/fill state. The new state applies to the primitives appended after the instruction. `ScalePen(False)` keeps line width visually constant when the inserted symbol is scaled; `True` scales the pen with the symbol.
`FontColor`, `FontHeight`, `FontScale`, `FontName`, `FontBold`, `FontItalic`, `FontUnderline`	Change the current text state for subsequent `Text` instructions. `FontScale` is limited internally to the range 0.4..2.0.
`Path(True)` ... `Path(False)`	Builds one compound path from intervening `Line`, `Polyline`, `Bezier` and `Arc` instructions. Closing the path applies the current fill and stroke once. Other figure and style calls are ignored while the path is open, so set its style before `Path(True)`.
`LinkPoint(X, Y)`	Adds a connector/snapping point to the symbol. Link points follow the instance translation, rotation and scale.
`SetNotDrawScale(Value)`	Fluent setter for `NotDrawScale`. It only bypasses the document drawing scale when the symbol is defined in `unMM`.

13.3 Useful definition, instance and library members

Class	Member	Developer notes
`TEDWLibSymbol`	`Name`, `Category`, `Note`, `Guid`	Definition metadata. Categories may be hierarchical by separating levels with `/`, for example `Furniture/Kitchen`. `Guid` is generated by `AddSymbol` and identifies the definition when a drawing is saved and loaded.
`TEDWLibSymbol`	`Units`, `InsertX`, `InsertY`, `NotDrawScale`	Coordinates passed to primitive methods use the definition units. Supported units are `unMM`, `unINCH`, `unCM`, `unDM`, `unM`, `unFt` and `unYd`. The insertion point is the placement and rotation reference.
`TEDWLibSymbol`	`ItemsCount`, `Item[Index]`, `NumLinks`, `Link[Index]`, bounds	Read the generated instruction list and connector points. Bounds are exposed as `BLeft`, `BTop`, `BRight` and `BBottom`; indexes for items and links are one-based.
`TEDWSymbol`	`Angle`, `Scale`, `Symbol`	Per-instance rotation (degrees), size multiplier and referenced definition. `Angle` accepts 0..360; `Scale` is clamped to 0.1..100. Assign the definition before assigning/using the instance points so its bounds can be calculated.

`TEDWSymbol`	`Guid`, `CenterPoint`, `InternalID`, `NumLinks`, `LinkPoint` / `LinkBase`	Inspect identity, insertion centre and transformed connector coordinates of the placed symbol.
`TEDWLibrary`	`AddSymbol`, `LoadFromFile`, `SaveToFile`, `FindSymbolByName`, `FindSymbolByGuid`, `CategoryList`	Create, persist and locate definitions in `.lyp` libraries. `CurrentSymbol` returns the last symbol found or added.
`TEmbedDrawVCL/FMX`	`LoadSymbolByName`, `LoadSymbolByGuid`, `CurrentLibSym`, `SymbolCount`, `Symbol[Index]`	Loads a library definition into the drawing. `TEDWFactory.AddSymbol(Owner, X, Y)` uses `CurrentLibSym`; the filename/name overload loads and inserts in one call and returns `nil` when the definition is not found.

A symbol definition used by a drawing is embedded in the drawing data. Keep symbol names unique inside a library and prefer the generated GUID when definitions can be renamed.

14.List of figures

1. Label editor 6
2. Flow chart 7
3. Technical drawing example 8
4. EmbedDraw VCL-FMX 3.2.0 9
5. Elements you can insert 10
6. New in version 3.1 radial, diameter and angle dimenion styles. 11
7. New in version 3.1 Image of map services in the image element or as the background of a layer. 12
8. New in version 3.1: TEDWRichText element to show formated text. 13
9. Mermaid Sequence diagram 16
10. Load GML files 18
11. Load GML files 21
12. Load GPX files 22
13. EmbedDraw VCL components 36
14. Define new variable 37
15. Search path variable 37
16. EmbedDraw FMX Components 38
17. Define FMX variable paths 39
18. Assign the search path variable 39
19. LanguageFileName property 41
20. Select Map dialog (Layers editor) 49
21. Dimension styles editor 53
22. VCL Layer editor 54
23. Select background image 54
24. Editor properties for emLabelling mode 55
25. Properties editor for drawing and flow chart mode 56
26. Title block 57
27. Preview 58
28. Inspector component 72
29. Select map image button 75
30. Editor 77
31. Align label topleft 81
32. Align label centered 81
33. Editor's settings dialog 83
34. Symbol defined as square with 1 x 1 m 90
35. Drawing properties, setting 1:10 scale 91
36. Map direction arrow 92
37. Scale and rotate symbol 93

15.List of tables

1. EmbedDraw VCL components 36
2. EmbedDraw VCL components 36
3. Line elements I 42
4. Line elements II 42
5. 2D Elements I 43
6. 2D Elements II 43
7. 2D Elements III 43
8. Text elements I 44
9. Text elements II 44
10. Symbolism 44
11. Dimension types I 45
12. Dimension types II 45
13. Dimension types III 45
14. Barcodes I 46
15. Barcode II 46
16. Package 46
17. Table element 47
18. Measure property 49
19. Options property 50
20. Settings property 51
21. Rulers properties 78
22. Editor's Grid property 78
23. SelectionColors property options 79
24. Snap options 79
25. Editor options 81