Developing UI controls for 3 frameworks, 3 IDEs and 5+ operating systems

Introduction

As a Pascal developer, we have a myriad of solutions and ways to develop applications these days. From classic Delphi VCL Windows applications to FireMonkey-based cross-platform applications for Windows, iOS, Android and macOS and using Lazarus/FPC that also gives us a route today to create applications for Linux and its many variants.

In all cases, our beloved Pascal language is common for all these ways but the framework for creating the UI differs. Delphi VCL for Windows, Delphi FMX for cross-platform and Lazarus LCL for cross-platform FPC compiler. At first sight, all 3 frameworks are internally quite different, especially VCL versus FMX or LCL versus FMX. This implies that so far, typical custom component development is done separately from VCL and FMX and in many cases also from LCL. In this article, we'll present the FNC-way to write a custom UI control once and use it in all 3 frameworks and for 5+ operating systems.

Challenges

There are several challenges though we'll like to explain first and knowing these challenges will give a better understanding of what FNC (**F**ramework **N**eutral **C**omponents) is and how it can be used.

A first challenge is that UI controls have a different class hierarchy in the different frameworks. In VCL, a custom control typically descends from TCustomControl defined in the VCL.Controls unit. In FMX, a custom control descends from TControl defined in the FMX.Controls unit. In LCL, the custom control descends from TCustomControl as well from the Controls unit.

When we want a single source component for 3 frameworks, we'd need 3 different class hierarchies. To install the components in the IDE, we'd also need a package dependency to VCL.DCP and FMX.DCP for Delphi and LCL.LPI for Lazarus. Hence, in Delphi we are forced to install our multi-framework component via two packages, one for VCL and one for FMX. And, as a unit can only be present in one package for install in the IDE, we'll need at least 2 component unit files. For LCL, this is less of an issue as it is a separate running IDE anyway.

Another challenge is the implementation of the base T(Custom)Control. There are numerous differences between the TControl/TCustomControl in VCL,FMX and LCL framework. In FMX, the coordinate system is floating point based. The method signatures for keyboard & mouse events are different. In VCL and LCL we deal with a TBrush & TPen to paint something while in FMX, this is done via a TFill and TStroke. The TCanvas in VCL and LCL are completely different from the FMX TCanvas. An image in VCL/LCL is handled via the TPicture base class and descendents TJPegImage, TPngImage, TGifImage, TBitmap while in FMX, there is only one TBitmap class that internally deals with JPEG, BMP, GIF, PNG file formats.

With so many differences & challenges, we'd already be inclined to give up.

Solutions

The TMS software team came up with the FNC (**F**ramework **N**eutral **C**omponents) abstraction layer that abstracts all differences between the frameworks and makes it possible to use a single API to write custom UI controls for the 3 frameworks VCL, FMX and LCL.

To start with, there are 3 new different base classes:

- **TTMSFNCCustomControl**: the base class for a custom control
- **TTMSFNCCustomScrollControl**: the base class for a custom control with scrollable surface and scrollbars
- **TTMSFNCCustomComponent**: the base class for a non-visual but framework dependent control

The base class introduces a framework neutral method interface for most common functionality:

```
TTMSFNCCustomControl = class

protected

  procedure HandleMouseLeave; virtual;

  procedure HandleMouseEnter; virtual;

  procedure HandleMouseDown({%H-}Button: TTMSFNCMouseButton; {%H-}Shift:
TShiftState; {%H-}X, {%H-}Y: Single); virtual;

  procedure HandleMouseMove({%H-}Shift: TShiftState; {%H-}X, {%H-}Y:
Single); virtual;

  procedure HandleDblClick({%H-}X, {%H-}Y: Single); virtual;

  procedure HandleMouseUp({%H-}Button: TTMSFNCMouseButton; {%H-}Shift:
TShiftState; {%H-}X, {%H-}Y: Single); virtual;

  procedure HandleKeyPress(var {%H-}Key: Char); virtual;

  procedure HandleKeyDown(var {%H-}Key: Word; {%H-}Shift: TShiftState);
virtual;

  procedure HandleDialogKey(var {%H-}Key: Word; {%H-}Shift: TShiftState);
virtual;

  procedure HandleKeyUp(var {%H-}Key: Word; {%H-}Shift: TShiftState);
virtual;

  procedure HandleMouseWheel({%H-}Shift: TShiftState; {%H-}WheelDelta:
Integer; var {%H-}Handled: Boolean); virtual;

  procedure HandleDragOver(const {%H-}Source: TObject; const {%H-}Point:
TPointF; var {%H-}Accept: Boolean); virtual;

  procedure HandleDragDrop(const {%H-}Source: TObject; const {%H-}Point:
TPointF); virtual;

  procedure Paint; virtual;

end;
```

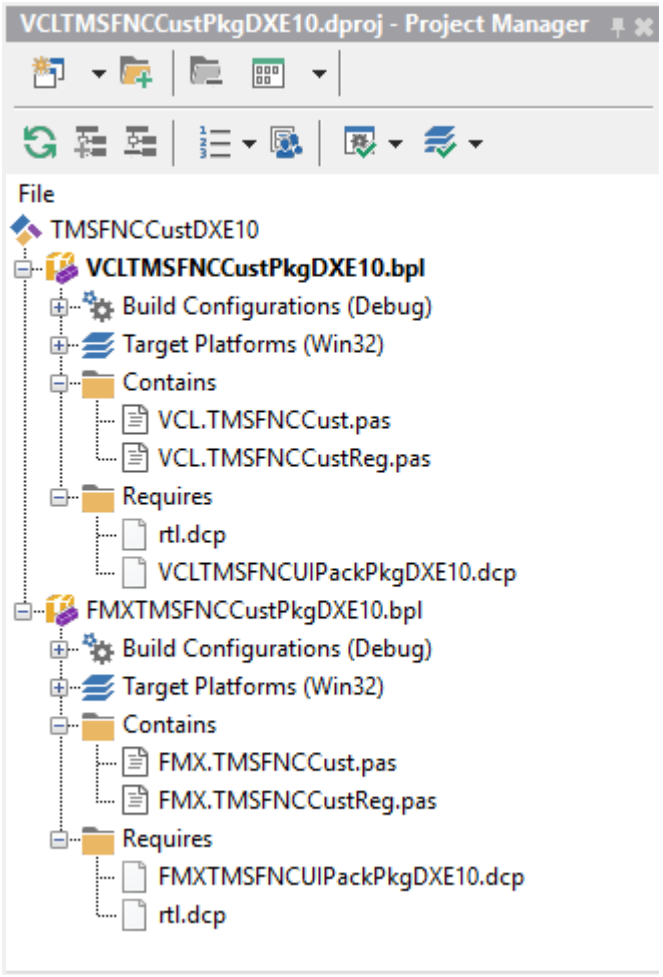
For handling all painting, the FNC layer offers several framework neutral classes and types:

- **TTMSFNCGraphics**: Canvas
- **TTMSFNCGraphicsStroke**: line style, width, color
- **TTMSFNCGraphicsFill**: solid, gradient fill
- **TTMSFNCGraphicsFont**: font with suppor for text color
- **TTMSFNCGraphicsColor**: gcRed, gcGreen, ...
- **TTMSFNCBitmap**: one class covering BMP,JPEG,PNG,GIF support

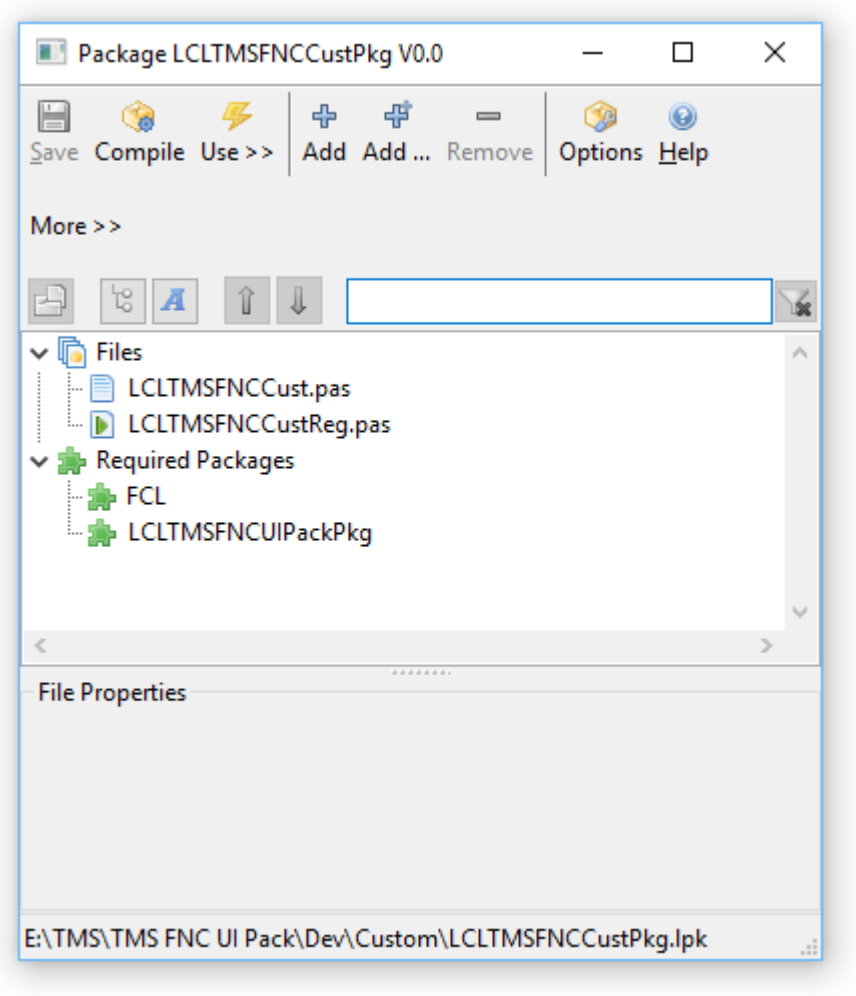
All this abstraction is the core of the TMS FNC UI Pack that you can find at:
http://www.tmssoftware.com/site/tmsfncuipack.asp

Getting to grips, defining packages for the first FNC custom UI control

We'll use 3 packages: a package for VCL, a package for FMX and a package for LCL. We can install the VCL & FMX package simultaneously in the Delphi IDE and the LCL package in the Lazarus IDE. The package for the custom control will have a dependency to the framework plus to the TMS FNC UI Pack that brings the abstraction framework. The structure of the packages in the Delphi 10.1 Berlin IDE is:



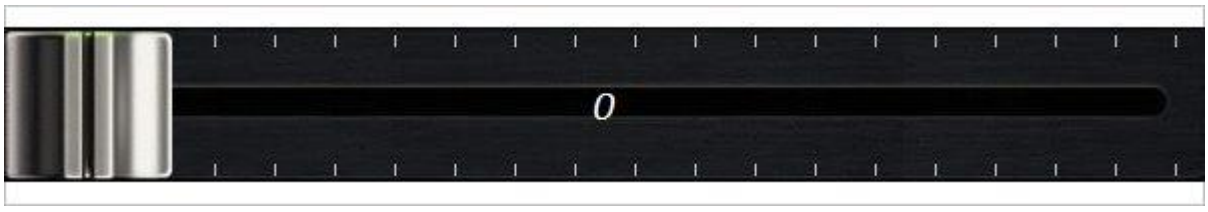
and in Lazarus, this is:



Time for writing code...

As we'll code against the FNC framework, we'll have a dependency on units VCL.TMSFNCCustomControl, VCL.TMSFNCGraphics for VCL, FMX.TMSFNCCustomControl, FMX.TMSFNCGraphics for FMX and LCLTMSFNCCustomControl, LCLTMSFNCGraphics units for LCL. We'll write the code once for the VCL framework and with a 2 line powershell script, we'll transpose the unit for FMX and LCL.

The control we want to present is a fancy trackbar. This is for several reasons. We want to create a control here that reacts to keyboard and mouse (touch) and a control that uses custom painting with images, text and lines.



The published interface for our fancy trackbar is:

```
type
  TTMSFNCFancyTrackbar = class(TTMSFNCCustomControl)
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property Font: TTMSFNCGraphicsFont read FFont write SetFont;
    property Min: integer read FMin write SetMin;
    property Max: integer read FMax write SetMax;
    property Slider: TTMSFNCBitmap read FSlider write SetSlider;
    property Thumb: TTMSFNCBitmap read FThumb write SetThumb;
    property TickStroke: TTMSFNCGraphicsStroke read FTickStroke write
SetTickStroke;
    property Ticks: integer read FTicks write SetTicks default 20;
    property Position: integer read FPosition write SetPosition default 0;
  end;
```

We have a minimum and maximum for the trackbar and a position. There is an image that holds the trackbar slider background and an image for the trackbar thumb. There is a font property to set the font to display the position and optional tickmarks with number of tickmarks and tickmark line style and color that can be set.

The protected section of TTMSFNCFancyTrackbar contains the overrides for handling keyboard, mouse and do the painting:

```
type
  TTMSFNCFancyTrackbar = class(TTMSFNCCustomControl)
  protected
    procedure HandleMouseDown({%H-}Button: TTMSFNCMouseButton; {%H-}Shift:
TShiftState; {%H-}X, {%H-}Y: Single); override;
    procedure HandleMouseMove({%H-}Shift: TShiftState; {%H-}X, {%H-}Y:
Single); override;
    procedure HandleMouseUp({%H-}Button: TTMSFNCMouseButton; {%H-}Shift:
TShiftState; {%H-}X, {%H-}Y: Single); override;
    procedure HandleKeyDown(var {%H-}Key: Word; {%H-}Shift: TShiftState);
override;
    procedure Draw(AGraphics: TTMSFNCGraphics; {%H-}ARect: TRectF);
override;
  end;
```

This control will handle the left & right arrow keys. It will handle mouse down, mouse move and mouse up for moving the thumb with the mouse or touch and the painting for the control happens via the override of the Draw() method.

Let's start with the implementation of the Draw() override, where the control is painted in the TTMSFNCGraphics context:

```
procedure TTMSFNCFancyTrackbar.Draw(AGraphics: TTMSFNCGraphics; ARect:
TRectF);
var
```

```
  i: integer;
  x,y,d,h: Single;
  pt1,pt2: TPointF;
begin
  inherited;

  // draw the control background
  AGraphics.Fill.Color := Color;
  AGraphics.DrawRectangle(ARect);

  y := 0;
  h := 0;

  // draw the trackbar slider background
  if Assigned(FSlider) then
  begin
    h := FSlider.Height;
    y := (Height - FSlider.Height) / 2;
    AGraphics.DrawBitmap(0, y, FSlider);
  end;

  d := 10;

  if (Ticks > 0) then
    d := Round(Width / Ticks);

  // draw the tickmarks above and below the slider
  for i := 0 to Ticks do
  begin
    AGraphics.Stroke.Assign(TickStroke);
    pt1 := PointF(i*d + (d / 2), y + 4);
    pt2 := PointF(i*d + (d / 2), y + 10);
    AGraphics.DrawLine(pt1,pt2);

    pt1 := PointF(i*d + (d / 2), y + h - 4);
    pt2 := PointF(i*d + (d / 2), y +h - 10);
    AGraphics.DrawLine(pt1,pt2);
  end;

  // draw the position value centered
  AGraphics.Font.Assign(Font);
  AGraphics.DrawText(ARect, IntToStr(Position), false, gtaCenter,
gtaCenter, gttNone);

  // draw the trackbar thumb at the x coordinate matching the Positon value
  if Assigned(FThumb) then
  begin
    y := (Height - FThumb.Height) / 2;
    x := 0;

    if (Max <> Min) and (Width - FThumb.Width > 0) then
      x := Round(Position / (Max - Min) * (Width - FThumb.Width));

    AGraphics.DrawBitmap(x,y,BitmapToDrawBitmap(FThumb));
  end;
end;
```

Key handling is quite easy. The override of HandleKeydown() checks for the left or right arrow and increments or decrements the Position accordingly:

```
procedure TTMSFNCFancyTrackbar.HandleKeyDown(var Key: Word; Shift:
TShiftState);
begin
  inherited;
  if Key = KEY_LEFT then
  begin
    Position := Position - 1;
  end;

  if Key = KEY_RIGHT then
  begin
    Position := Position + 1;
  end;
end;
```

For mouse handling, there is a simplified implementation that tracks in the mouse down whether the mouse is in the thumb area and if so, sets a flag (FInThumb private boolean variable). This flag is used in the mouse move override to move the thumb and finally, the flag is cleared in the mouse up handler:

```
procedure TTMSFNCFancyTrackbar.HandleMouseDown(Button: TTMSFNCMouseButton;
  Shift: TShiftState; X, Y: Single);
var
  h,w: Single;
begin
  inherited;
  SetFocus;

  FInThumb := false;

  if Assigned(FThumb) then
  begin
    h := (Height - FThumb.Height) / 2;

    if (Y > h) and (Y < h + FThumb.Height) then
    begin

      w := Round(Position / (Max - Min) * (Width - FThumb.Width));

      if (x > w) and (x < w + FThumb.Width) then
      begin
        FInThumb := true;
        CaptureEx;
      end;
    end;
  end;
```

```
end;

procedure TTMSFNCFancyTrackbar.HandleMouseMove(Shift: TShiftState; X,
  Y: Single);
begin
  inherited;

  if FInThumb then
  begin
    if (Max <> Min) then
      Position := Round((x / (Width - FThumb.Width)) * (Max - Min));
  end;
end;

procedure TTMSFNCFancyTrackbar.HandleMouseUp(Button: TTMSFNCMouseButton;
  Shift: TShiftState; X, Y: Single);
begin
  inherited;
  if FInThumb then
    ReleaseCaptureEx;
  FInThumb := false;
end;
```

Other than some property setter methods, there is not much more to it to write the control.

Transposing to FMX and LCL

All there is left to do is transpose the source we wrote for VCL to FMX and LCL. All we do here is create new units with VCL. replaced by FMX. and VCL. by LCL respectively:

powershell -command "(gc VCL.TMSFNCCust.pas) -replace 'VCL.TMS','LCLTMS' |Out-file LCLTMSFNCCust.pas -Encoding utf8"

powershell -command "(gc VCL.TMSFNCCust.pas) -replace 'VCL.TMS','FMX.TMS' |Out-file FMX.TMSFNCCust.pas -Encoding utf8"

We can now build our packages using FMX.TMSFNCCust.pas and LCLTMSFNCCust.pas and start using the control.

Using the FNC UI control with a single source base

FNC not only brings the benefit we can write the code for a UI control once, it also brings the benefit that the code to use the control from the different frameworks will be 100% identical.

This code snippet programmatically creates a TTMSFNCFancyTrackBar in code, loads the thumb & slider image and sets the font to use to show the position as a number:

```
begin
```

```
  tb := TTMSFNCFancyTrackBar.Create(Self);
  tb.Color := gcWhite;
  tb.Width := 600;
  tb.Height := 100;
  tb.Top := 100;
  tb.Left := 100;
  tb.Parent := Self;

  tb.Thumb.LoadFromFile(TTMSFNCUtil.GetAppPath + 'thumb.png');
  tb.Slider.LoadFromFile(TTMSFNCUtil.GetAppPath + 'slider.png');

  tb.Font.Color := gcWhite;
  tb.Font.Size := 16;
  tb.Font.Style := [TFontStyle.fsItalic];
  tb.Font.Name := 'Calibri';
end;
```

Conclusion

With the TMS FNC UI Pack, the TMS software team brings a set of powerful and complex UI controls you're free to use in the IDE and framework of choice. With this article, we wanted to show that when using TMS FNC UI Pack, you're not restricted to the controls that we include but if you have a need for custom controls, it is really simple to create these. Especially the TTMSFNCGraphics graphical context class is a powerful abstract wrapper that can accelerate and make your graphics programming more powerful & faster even if you only need to target one single framework.

We hope you enjoy this new freedom to easily mix & change frameworks and IDEs and we are eager to learn what kind of exciting UI controls you created for FNC. At the same time, we're looking forward to your feedback and wishes for new FNC components or new features in our FNC controls