TMS Instrumentation Workshop for FireMonkey

Delphi XE2 offers cross platform support but in doing so, it requires that the user interface of applications is developed with an entirely new framework, FireMonkey, replacing the VCL. The FireMonkey framework is significantly different from the VCL framework. In this article, we'll touch the main differences between FireMonkey and the VCL. From there, we'll provide the basics of user interface component design with the FireMonkey framework and we'll give an overview of the first user interface components product for FireMonkey from TMS software: TMS Instrumentation Workshop for FireMonkey.

From VCL to FireMonkey

The VCL was designed as an object oriented framework for the Microsoft Windows operating system. With respect to user interface controls, this means a close relationship to the inner workings of the Windows API. This means basically means:

- based on window handles
- perform painting in a Windows device context
- send and receive Windows messages
- call Windows API functions

A typical example is a VCL TListBox. This is an object oriented wrapper around the Windows operating system LISTBOX class. It will use the window handle of the listbox, it can handle the different WM_* messages that a LISTBOX can receive in response to user interface actions, it can use the device context handle HDC to perform custom painting of items, it can call Windows API functions like GetSystemMetrics and it can send window messages like LB_ADDSTRING.

A typical complex user interface VCL component will use all these aspects of Windows programming, albeit within the object oriented skeleton of Object Pascal.
Enter the FireMonkey framework, and things are dramatically different. A first big change is that user interface controls will not map on the operating system defined controls. Controls are not longer based on a window handle, the device context does not exist anymore and there are no more Windows messages to send and receive. Instead, you will find a TCanvas and styles.

The TCanvas is a class that abstracts the painting of controls. Dependent on the operating system the application is running on, the FireMonkey TCanvas can use one of following underlying graphics technologies to perform the actual drawing: GDI+ on Windows XP and Windows Vista, Direct2D on Windows 7, Quartz on Mac OS-X and iOS or for 3D, DirectX on Windows and OpenGL on Mac OS-X and iOS.

So, by using the FireMonkey TCanvas, we can write code that draws lines, text, etc... and the underlying framework will call the proper graphics API dependent on the operating system to perform the actual drawing.

The Styles in the FireMonkey framework can be considered as a hierarchy of objects that make up a control. Objects can be as simple as a FireMonkey TRectangle or TLine but also something like a TButton, TScrollBar. A style can be described in a text format that is almost identical to the DFM file format but a style can also be dynamically created in code.

If we'd go back to the example of the TListBox in the VCL, the equivalent FireMonkey TListBox is built-up in an entirely different way. It will roughly consist of a TLayout which is a rectangle and in this rectangle, there is TRectangle that is the area for the items or container for other objects and right and at the bottom, TScrollBar controls will be used. Take in account that the TScrollBar is completely unrelated to the Windows SCROLLBAR class. The FireMonkey

TScrollBar is in itself built-up from an up and down TButton and rectangles to create the track area and track bar.

One of the powerfull features of the FireMonkey framework is that we can separately make up the design of a control by editing its style as DFM. We do not have to write any code. A bit similar to writing XAML or to write HTML, we can already create complex controls having many elements pure as textual description. On top of this, as the style is separated from the code, we can change all colors, gradients, strokes independently from code or even replace a style description with another one without changing the code. This mechanism provides us the capability to have a style with color settings that match a typical user interface for Windows and another style with color settings matching Mac OS-X etc.. By simply replacing the style, the user interface adapts automatically and the code behind remains the same.

A basic stylable FireMonkey component

The class for the basic stylable FireMonkey component is:

```
{$R CustomStyledControl.win.res}

type
  TCustomStyledControl = class(TStyledControl)
  protected
    function LoadFromResource: TControl;
    function GetStyleObject: TControl; override;
  end;
```

The file CustomStyledControl.win.res is generated from the .RC file CustomStyledControl.win.rc with:
CustomStyledControlStyle RCDATA "CustomStyledControl.win.style" and the actual style file content:

```
CustomStyledControl.win.style:
  object TLayout
    Position.Point = '(72,96)'
    Width = 377.000000000000000000
    Height = 265.000000000000000000
    object TRectangle
      Position.Point = '(16,8)'
      Width = 241.000000000000000000
      Height = 193.000000000000000000
    end
    object TScrollBar
      Position.Point = '(24,200)'
      Width = 233.000000000000000000
      Height = 18.000000000000000000
      TabOrder = 1
      Max = 100.000000000000000000
      Orientation = orHorizontal
      SmallChange = 1.000000000000000000
    end
    object TScrollBar
      Position.Point = '(232,-200)'
      Width = 18.000000000000000000
      Height = 150.000000000000000000
      TabOrder = 1
      Max = 100.000000000000000000
      Orientation = orVertical
      SmallChange = 1.000000000000000000
    end
    object TButton
      Position.Point = '(256,200)'
      Width = 17.000000000000000000
      Height = 22.000000000000000000
      TabOrder = 2
      StaysPressed = False
      IsPressed = False
      Text = '?'
      StyleName = 'btn'
    end
  end
```

The code consists of an override of the GetStyleObject method. This method is called whenever the FireMonkey framework needs access to the style of a control. This style can already exist or have been customized in the StyleBook and by default, when available, the FireMonkey framework will retrieve the style from the StyleBook. When no style is present in the StyleBook, the component is assumed to return the style itself and in this case, the style will be loaded from a resource that holds a textual description of the objects that make up the style:

```
function TCustomStyledControl.GetStyleObject: TControl;
var
  obj: TControl;
begin
  obj := inherited GetStyleObject;

  // not found in default or custom style book, so create from resource
  // dynamic creation of TLayout instance or other FMX classes is alternative
  if not Assigned(obj) then
    obj := LoadFromResource;

  Result := obj;
end;

function TCustomStyledControl.LoadFromResource: TControl;
```

```
var
  S: TResourceStream;
  str: String;

begin
  Result := nil;

  // create resource name from class name
  str := ClassName + 'style';
  Delete(str, 1, 1);

  if FindRCData(HInstance, str) then
  begin
    // load from RT_RCDATA resource type
    S := TResourceStream.Create(HInstance, str, RT_RCDATA);
    try
      Result := TControl(CreateObjectFromStream(nil, S));
    finally
      S.Free;
    end;
  end;
end;
```

The StyleName and the style-contract

In most cases, interaction is needed between the control's code and the style. This can be in two ways:

- the control might need to change a property of a style object
- the control might want to get notified when an interaction with an object in the style happens

To do this, each object in the style can be given a name. In the style definition above, you can see that the StyleName for the TButton object is set as 'btn'. In The FireMonkey framework, we can use the function FindStyleResource to retrieve the style object in the control's style with a given name. We can then get access to this style object and for example change it's Text property or attach an event handler to the OnClick event:

```
procedure TCustomStyledControl.UpdateText;
var
  fo: TFMXObject;
begin
  // btn is part of the style contract, used to identify style object
  // where text should be updated.
 fo := FindStyleResource('btn');
  // verify the style element effectively exists in the style and apply
  if Assigned(fo) and (fo is TButton) then
  begin
    // update the property of the style object
   (fo as TButton).Text := FText;
    // assign an event handler for the OnClick event
    (fo as TButton).OnClick := ButtonClick;
  end;
end;
```

If we'd want to change the style of the control without changing the actual code and we'd want that the control can still update some text or attach an event handler for OnClick, we'd need to make sure that at least a style object with StyleName 'btn' and of the type TButton or a descending class exists in the style. This requirement of the control with respect to the style is called the style-contract. When replacing control's styles, changing control's styles, the style-contract should always be respected.

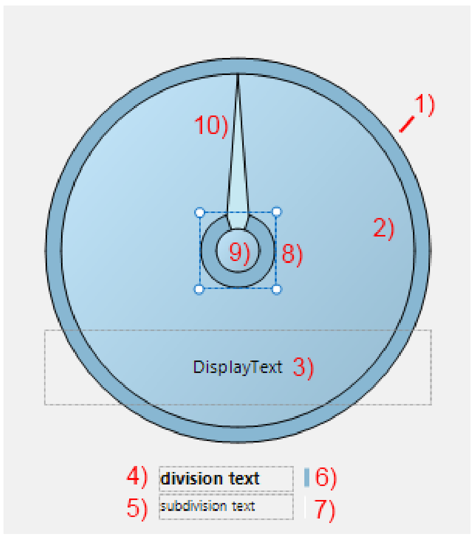
The TMS Instrumentation Workshop for FireMonkey

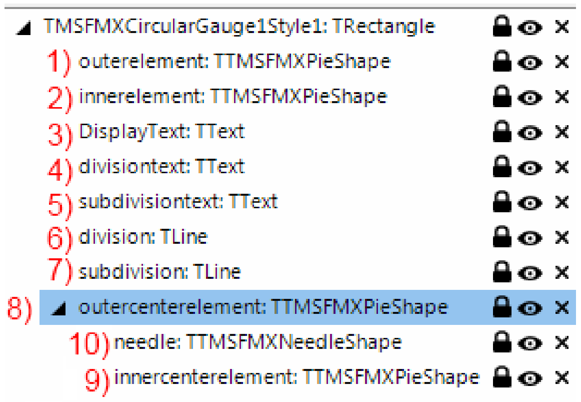
The first component set for FireMonkey cross-platform development we created at TMS software was entirely created with the style-ability in mind. All components were architected from the ground-up to fit in the design philosophy of the FireMonkey framework. Wherever possible, styles are extensively used. Developed for creating instrumentation and multimedia applications, you'll find gauges, meters, a scope, LED controls and more.

Gauges

In the category of gauges, 3 components are provided: TTMSFMXCircularGauge, TTMSFMXLinearGauge and TTMSFMXJogMeter. The TTMSFMXCircularGauge and TTMSFMXLinearGauge share a common architecture. The gauges can display one or more needles from a minimum to a maximum value. Within this minimum/maximum, one or more sections can be added. A section is an area on the gauge, indicating a range of values and for which a specific color/gradient can be set. At specific values on the gauge, SetPoints can be added and different shapes/colors can be selected as SetPoint.

Following style objects make up the appearance of the TMSFMXCircularGauge:



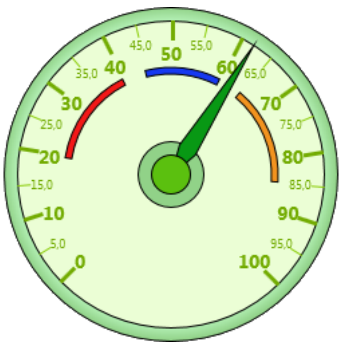


◢ TMSFMXCircularGauge1Style1: TRectangle
- 1) outerelement: TTMSFMXPieShape
- 2) innerelement: TTMSFMXPieShape
- 3) DisplayText: TText
- 4) divisiontext: TText
- 5) subdivisiontext: TText
- 6) division: TLine
- 7) subdivision: TLine
- 8) ◢ outercenterelement: TTMSFMXPieShape
  - 10) needle: TTMSFMXNeedleShape
  - 9) innercenterelement: TTMSFMXPieShape

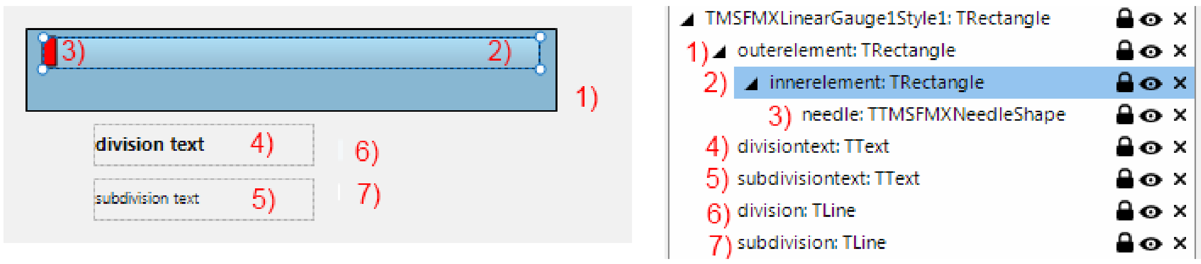
1) The outer background object of the gauge.
2) The inner object of the gauge. This object holds the sections, needles, setpoints, divisions and subdivisions.
3) The displaytext of the gauge, this can be set on component level.
4) 5) The Division and SubDivision text appearance.
6) 7) The Division and SubDivision line appearance.
8) The outer center object.
9) The needle.
10) The inner center object which holds the needle.

Via the IDE style editor, the properties that control the appearance of these style objects can be changed. In the IDE style editor, you can see in the list of style objects the style name and the object class. It is possible to add or even replace style objects within this style.

The end result after customization of colors and adding three sections could be:

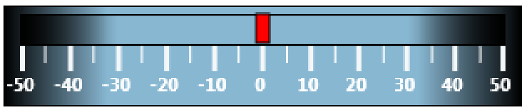


The linear gauge, TTMSFMXLinearGauge is similar to the circular gauge. It also offers Sections, SetPoints, one or more needles and configurable division lines and division values. These return in the IDE style editor for configuration:



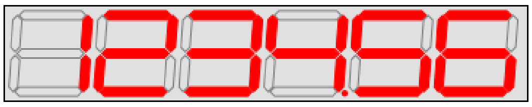
TMSFMXLinearGauge1Style1: TRectangle
1) outerelement: TRectangle
2) innerelement: TRectangle
3) needle: TTMSFMXNeedleShape
4) divisiontext: TText
5) subdivisiontext: TText
6) division: TLine
7) subdivision: TLine

The jogmeter is a similar component as the TTMSFMXLinearGauge but has no minimum and maximum values. The range is defined via a combination of Divisions, Aperture and Step. The needle always remains in the center of the control and the displayed range changes if the value is set.

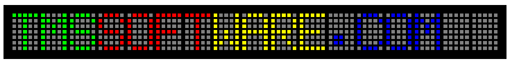


LED labels, bars, meters

TTMSFMX7SegLED is a control to display a value via 7 segment LEDs. The number of digits and the number of decimals can be set. Also here, it is possible to change the appearance of the 7 segment LEDs via the IDE style editor. The basic object defining the appearance of one 7 segment LED is a TTMSFMX7SegLEDShape. This object has properties Fill and FillActive that set the colors of the LED when it is in on and off state.



As a TTMSFMX7SegLED is limited to displaying numbers, the TTMSFMXMatrixLabel can display text. The matrix label has only one stylable element, the LEDText shape itself. This has properties to change the amount of LEDs, the size of the LEDs and the direction. The Text that is displayed inside the matrixlabel is set with the Text property in the Object Inspector. The matrixlabel has the capability to autoscroll the text from right to left or vice versa. This is set with the ScrollDirection property.



Parts of the text can have different colors. This can be done by using a '%' character followed by a hex number from 0-F. This will change the color to a predefined set of colors. The color remains applied until a new color code is set in the text.

Example:

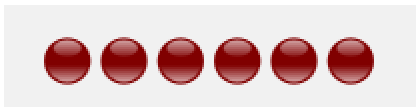
```
TMSFMXMatrixLabel1.Text := 'tms%Asoft%Cware%5.com'
```

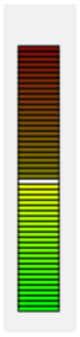
with %A = red, %C = yellow, %5 = blue.

Another variation of the LED controls is the TTMSFMXLEDBar and TTMSFMXLEDMeter.

The TTMSFMXLEDBar is a set of discrete LEDs and dependent on the Value of the TTMSFMXLEDBar, the number of LEDs in on state is set. In the TTMSFMXLEDBar each LED can have the same or a different color for the on and off state.

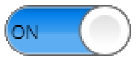
TTMSFMXLEDMeter is a bar of rectangular LEDs and the colors of each LED are automatically set ranging from a start color to an end color. The Value property of the TTMSFMXLEDMeter determines the number of rectangular LEDs in on state. In addition, peak value indication is possible in this control.



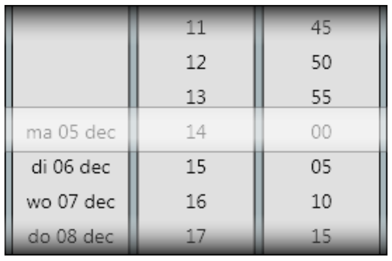


Slider and spinner

The TTMSFMXSlider is an on-off button similar to the on-off button in iOS. It has two states and 4 style objects that make up the appearance of the control in its two states. Via the IDE style editor, this can be entirely customized.



TTMSFMXSpinner is a scrolling selector control and can also be found in iOS. It can be used to select a date or time but is versatile enough to select other types of data. The spinner has a Columns collection property, and when adding columns, they become available in the IDE style editor. Each column can be styled separately and can show a different range of items. Just like in iOS, the spinner wheels offer smooth scrolling with inertia.

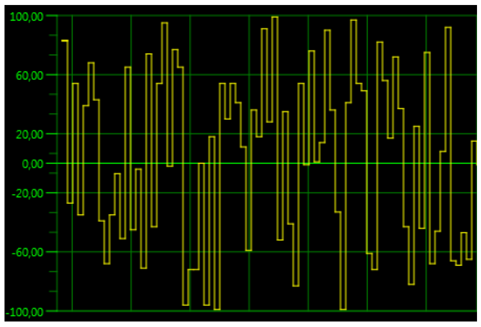


Scope

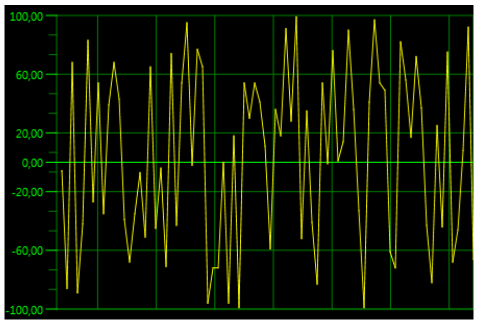
The TTMSFMXScope has the capability of displaying values of multiple channels with a certain interval and frequency. The scope has a channels collection property that can be used to add channels. Each channel has a Color, Font, ShowValue, Style, ValueFormat, Visible and Width property. The Style property is used to switch between line and bar styles. Data can be added to the scope in two ways. In AutoUpdate mode, the scope triggers the event OnNeedData requesting the data at that time for a given channel. The data is returned via the var parameter of this event. Alternatively, the data can be set by calling the methods TTMSFMXScope.UpdateData(ChannelIndex, Value) and when the data is set this way for all channels, it is added to the scope with one call:

TTMSFMXScope.AddData.

Bar style



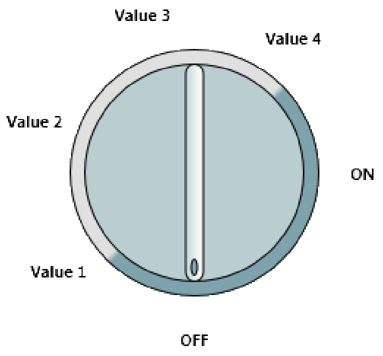
Line style

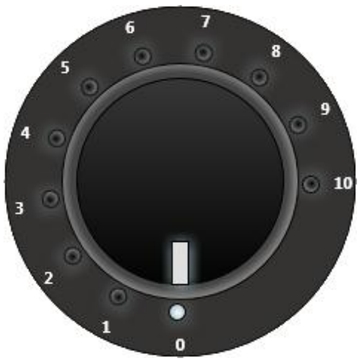


Switches

Finally, two switch controls are available: TTMSFMXRotarySwitch and TTMSFMXKnobSwitch. The TTMSFMXRotarySwitch is a basic switch, the TTMSFMXKnobSwitch offers LEDs in the outer circle that indicate in what position the switch is.

The values between with the switch control can select are set via the Positions collection. Each position has a Text property and a Tag property. For each position there is a value added on the switch. The position of the switch can be easily get or set via TTMSFMXRotarySwitch.Value. The first position value is 0, the second position 1, etc…





Conclusion

The FireMonkey framework uses a radically different approach from the VCL to create user interface controls. The FireMonkey TCanvas abstracts all the complexities of dealing with different graphics APIs on different operating systems. The mechanism to work with styles not only enables to separate code from the design of the visual appearance but also offers to customize, extend, adapt user interface controls in unprecedented levels as long as a style-contract is respected. The TMS Instrumentation Workshop is the first set of user interface components for FireMonkey from TMS software fully respecting this new design philosophy and architecture of FireMonkey. You can download a fully functional trial version of TMS Instrumentation Workshop for FireMonkey at http://www.tmssoftware.com/site/tmsfmxiw.asp. The registered version comes with full source code. The components can be used in applications that can be deployed to Win32, Win64, Mac OS-X and iOS devices iPhone, iPad, iPod.