

TMS FNC UI Controls, a set of framework neutral controls

Introduction

As your customers use an ever increasing number of devices & operating systems, it is a daily challenge for us, software developers to make our software available for the myriad of target platforms in use. Fortunately, as Pascal developers, we already have a huge benefit that Delphi meanwhile targets 4 platforms: Windows, Mac OSX, iOS and Android. Add Lazarus to that, and Pascal developers can also target Linux and its derivatives such as Raspbian and many more. The 3 main frameworks we have available to create our software for these platforms are: VCL, FMX and LCL. The framework to use will be determined by the target and the IDE used. That implies that the controls that can be used are also typically determined by this choice and might limit your abilities when some controls or control features are not available for one of these frameworks. Addressing that limitation is exactly one of the primary goals of the TMS FNC UI Controls. It offers you a set of powerful & feature-rich UI controls that you can use in Delphi's VCL framework, FMX framework and Lazarus LCL framework. It allows to create Win32, Win64, Mac OS-X, iOS, Android, Linux, Raspbian, ... applications with a single codebase and a single learning curve.

Concept

FNC controls enable you to write and use 100% identical Pascal code, be it in a VCL app, FMX app or LCL app. The FNC components methods, properties and events are therefore 100% identical regardless of the framework being used. As an example, the following code creates a new event in our FNC Planner control:

```
var
  plIt: TTMSFNCPlannerItem;

begin
  plIt := TMSFNCPlanner1.Items.Add;
  plIt.StartTime := Date + EncodeTime(14,0,0,0);
  plIt.EndTime := Date + EncodeTime(16,0,0,0);
  plIt.Title := 'New event';
  plIt.Text := 'Content';
end;
```

and from this code, it is impossible to tell whether it will be from a VCL, FMX or LCL app. In the application UI, it will also look exactly the same regardless of framework or operating system:

	Tuesday	Wed
7 00		
30		
8 00		
30	New event	
9 00	Content	
30		
10 00		
30		
11 00		
30		

This means that if you properly separate your logic or adopt an MVC approach, you can easily share .PAS files between VCL and FMX projects, between VCL and LCL projects etc... There are in this respect actually only two limitations. First limitation is the design-time form fileformat that is different between VCL, FMX and LCL. VCL uses the .dfm file, FMX uses the .fmx file and LCL uses the .lfm file. For applications for different devices with different form factors, it typically already requires to design the form separately for separate frameworks, so this isn't too much of a limitation. For other applications, a solution is to create the controls at runtime. A second limitation is the namespaces (unit names). To be able to register identical classnames for different framework controls in Delphi, it is required that these live in different namespaces. As such, the FNC VCL controls unit names have the prefix VCL., the FNC FMX controls unit names have the prefix FMX. and the FNC LCL controls use prefix LCL (without dot, to be able to support FPC versions older than v3.0)

In practice, this means that for the example above with the TMSFNCPlanner, the unit clauses for the different frameworks would be as below. To keep using a single source file, a solution is to set a define at project level depending on the framework and write:

```
uses
{$IFDEF VCL}
  VCL.TMSFNCPlannerBase, VCL.TMSFNCPlannerData, VCL.TMSFNCPlanner,
  VCL.TMSFNCCustomControl;
{$ENDIF}

{$IFDEF FMX}
  FMX.TMSFNCPlannerBase, FMX.TMSFNCPlannerData, FMX.TMSFNCPlanner,
  FMX.TMSFNCCustomControl;
{$ENDIF}

{$IFDEF LCL}
  LCLTMSFNCPlannerBase, LCLTMSFNCPlannerData, LCLTMSFNCPlanner,
  LCLTMSFNCCustomControl;
{$ENDIF}
```

In the same way, when used, we could include the resource of our form file that is different in each framework via a conditional define:

```
{ $IFDEF VCL }
{ $R *.dfm }
{ $ENDIF }

{ $IFDEF FMX }
{ $R *.fmx }
{ $ENDIF }

{ $IFDEF LCL }
{ $R *.lfm }
{ $ENDIF }
```

These are of course the things you need to take in account when you want to create a single codebase to build projects with multiple frameworks. In other cases, you do not need to take care of this and you can enjoy the exact same feature set of this UI component library irrespective of the IDE and platform you target.

Another important core concept is the introduction of the TMS FNC Graphics library that is included. This enables you to write graphics code that is framework independent. This includes framework neutral colors, fill, stroke, alignment, font, path types and the `TTMSFNCGraphics` class using this to draw everything you need. This is a sample code snippet of framework neutral drawing:

```
var
  gr: TTMSFNCGraphics;

begin
  gr := TTMSFNCGraphics.Create(PaintBox1.Canvas);
  gr.Fill.Color := gcYellow;
  gr.Stroke.Color := gcGray;
  gr.DrawRectangle(0,0,100,20);
  gr.Font.Color := gcRed;
  gr.DrawText(2,0,100,20,'Hello world',false)
  gr.Free;
end;
```

The result is:

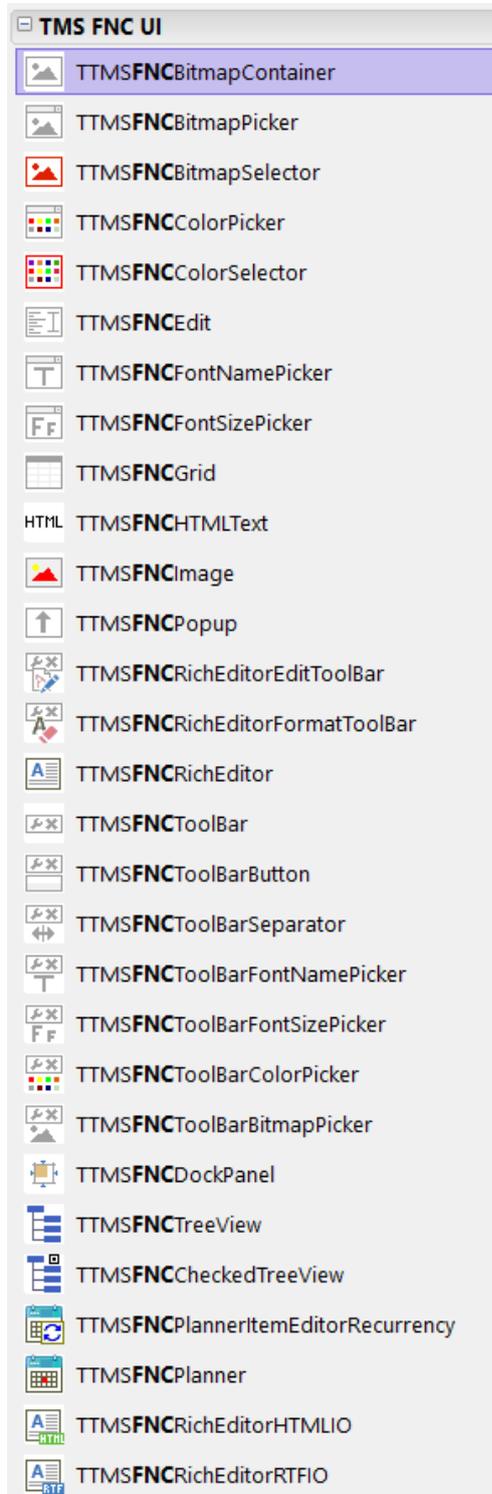


and is exactly the same on every framework, target, device, ...

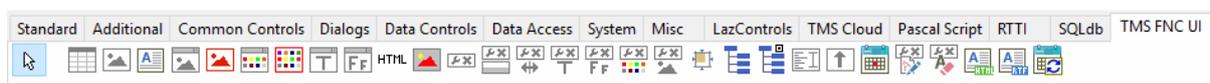
Controls

In TMS FNC UI Pack v1.0, there are already 29 controls included. On the tool palette this looks like:

VCL, FMX



LCL



This includes a grid, planner, richeditor, treeview, various color, font, fontsize, bitmap ... picker , toolbar, ... and more.

Introducing the TMS FNC Grid

The TMS FNC Grid is a high-performance, not data-bound grid capable of dealing with hundreds of thousands of rows, has a wide range of cell types and inplace editors, offers built-in sorting, filtering and grouping and can import and export data in several file formats. To illustrate some of the capabilities of the TMS FNC Grid,

```
begin
  TMSFNCGrid1.FixedRows := 2;
  TMSFNCGrid1.ColumnCount := 7;
  TMSFNCGrid1.MergeCells(1,0,2,1);
  TMSFNCGrid1.MergeCells(3,0,2,1);
  TMSFNCGrid1.MergeCells(5,0,2,1);

  TMSFNCGrid1.Cells[1,0] := 'Monday';
  TMSFNCGrid1.HorzAlignments[1,0] := gtaCenter;
  TMSFNCGrid1.Cells[1,1] := 'AM';
  TMSFNCGrid1.Cells[2,1] := 'PM';

  TMSFNCGrid1.Cells[3,0] := 'Tuesday';
  TMSFNCGrid1.HorzAlignments[3,0] := gtaCenter;
  TMSFNCGrid1.Cells[3,1] := 'AM';
  TMSFNCGrid1.Cells[4,1] := 'PM';

  TMSFNCGrid1.Cells[5,0] := 'Wednesday';
  TMSFNCGrid1.HorzAlignments[5,0] := gtaCenter;
  TMSFNCGrid1.Cells[5,1] := 'AM';
  TMSFNCGrid1.Cells[6,1] := 'PM';
  TMSFNCGrid1.AutoNumberCol(0);

  TMSFNCGrid1.AddCheckBox(1,2,false);
  TMSFNCGrid1.AddRadioButton(1,3,1);

  TMSFNCGrid1.AddProgressBar(3,2,50);
  TMSFNCGrid1.Cells[3,3] := 'Hello <B>world</B>';

  TMSFNCGrid1.AddBitmapFile(5,2,'e:\tms\calendar.png');
  TMSFNCGrid1.AddBitmapFile(5,3,'e:\tms\mail.png');

  TMSFNCGrid1.Cells[1,4] := 'Red';
  TMSFNCGrid1.Colors[1,4] := gcRed;
  TMSFNCGrid1.Cells[3,4] := 'Yellow';
  TMSFNCGrid1.Colors[3,4] := gcYellow;
  TMSFNCGrid1.Cells[5,4] := 'Lime';
  TMSFNCGrid1.Colors[5,4] := gcLime;

  TMSFNCGrid1.FontNames[1,4] := 'Courier New';
  TMSFNCGrid1.FontStyles[3,4] := [fsBold];
  TMSFNCGrid1.FontSizes[5,4] := 12;

  TMSFNCGrid1.AddNode(2,2);
end;
```

0	Monday		Tuesday		Wednesday	
1	AM	PM	AM	PM	AM	PM
2	<input type="checkbox"/>		50%			
3	<input type="radio"/>		Hello world			
4	Red		Yellow		Lime	
5						
6						
7						
8						
9						

As another quick introduction to the grid, this 2 line snippet demonstrates how data from a CSV file can be loaded and automatic filtering via a drop down in the column header is enabled:

```
TMSFNCGrid1.LoadFromCSV('e:\tms\cars.csv');
TMSFNCGrid1.Options.Filtering.DropDown := true;
```

Alfa Romeo	Honda	98	88	4	120	699000
Alfa Romeo	JAGUAR	74	106	4	144	769000
Alfa Romeo	LAMBORGHINI	70	114	4	155	899000
Alfa Romeo	Lexus	92	140	6	190	1099000
Alfa Romeo	LOTUS	70	114	4	155	1100000
Alfa Romeo	MASERATI					
Alfa Romeo	Mazda					
Alfa Romeo	MERCEDES					
Alfa Romeo	MG					
Alfa Romeo	166 2,0V6	1996	151	6	190	1350000
Alfa Romeo	166 2,5V6	2492	140	6	190	1460000
Alfa Romeo	166 3,0V6	2959	166	6	226	1580000
Alfa Romeo	Spider 1,8	1747	106	4	144	999000
Alfa Romeo	Spider 2,0	1970	114	4	155	1072000
Alfa Romeo	Spider 3,0	2959	141	6	192	1437000

Of course, this is just a very brief introduction to the TMS FNC Grid. Just the FNC grid alone could deserve multiple articles to cover it in detail. You can familiarize yourself with the TMS FNC Grid by reading the full PDF developers guide you can find at <http://www.tmssoftware.biz/download/manuals/TMSFNCGridDevGuide.pdf> or use the trial or full version of the component that comes with several samples.

Introducing the TMS FNC Planner

Our TMS FNC Planner is a scheduling component with various built-in time-axis options, i.e. a day, week, month, period, half-day period, timeline as well as custom time-axis mode where you can fully control the duration of each timeslot in the Planner. The Planner supports single and multi-resource views and can have the time-axis horizontal or vertical. When targeting the Planner to a mobile device, it will automatically use a touch-friendly approach to select, insert, delete, pan in the Planner. As a brief introduction to the TMS FNC Planner, we'll demonstrate a monthly car rental Planner with horizontal time axis and several resources in the left axis.

In this code, we set the time axis programmatically horizontal and add 9 resources with 3 groups of 3 resources, representing small, medium and large cars. Via TMSFNCPlanner.Mode, we set the time line to a day period timeline and via TMSFNCPlanner.ModeSettings the period is set to the month May. After some further customization of timeline size and font, some random car rentals are added on the Planner:

```
var
  grp: TTMSFNCPlannerGroup;
  plIt: TTMSFNCPlannerItem;
  i,d: integer;

begin
  TMSFNCPlanner1.OrientationMode := pomHorizontal;
  TMSFNCPlanner1.Mode := pmDayPeriod;

  TMSFNCPlanner1.Positions.Count := 9;
  TMSFNCPlanner1.Groups.Clear;
  grp := TMSFNCPlanner1.Groups.Add;
  grp.StartPosition := 0;
  grp.EndPosition := 2;
  grp.Text := 'Small';

  grp := TMSFNCPlanner1.Groups.Add;
  grp.StartPosition := 3;
  grp.EndPosition := 5;
  grp.Text := 'Medium';

  grp := TMSFNCPlanner1.Groups.Add;
  grp.StartPosition := 6;
  grp.EndPosition := 8;
  grp.Text := 'Large';

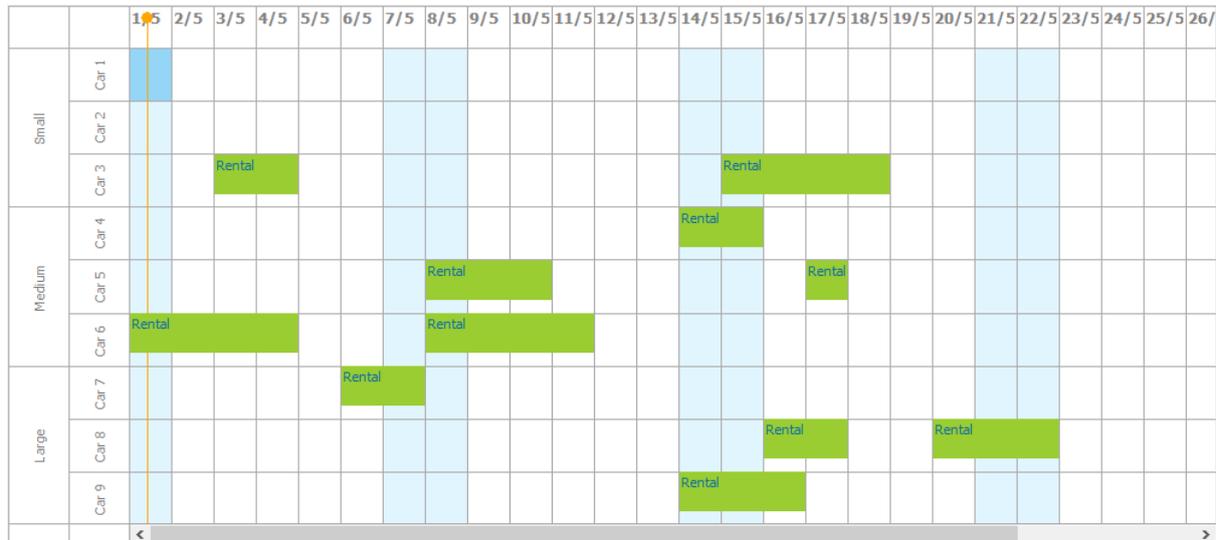
  TMSFNCPlanner1.Resources.Clear;
  for i := 0 to 8 do
  begin
    TMSFNCPlanner1.Resources.Add;
    TMSFNCPlanner1.Resources.Items[i].Text := 'Car '+inttostr(i + 1);
  end;

  TMSFNCPlanner1.TimeLineAppearance.LeftSize := 35;
  TMSFNCPlanner1.TimeLineAppearance.LeftFont.Size := 10;
  TMSFNCPlanner1.TimeLineAppearance.LeftFont.Style := [fsBold];
  TMSFNCPlanner1.ModeSettings.StartTime := EncodeDate(2016,5,1);
  TMSFNCPlanner1.ModeSettings.EndTime := EncodeDate(2016,6,1);
  TMSFNCPlanner1.TimeLine.DisplayUnitSize := 35;
  TMSFNCPlanner1.TimeLine.DisplayUnitFormat := 'd/m';
```

```

for i := 0 to 10 do
begin
  d := random(20);
  plIt := TMSFNCPlanner1.Items.Add;
  plIt.StartTime := EncodeDate(2016,5,1) + d;
  plIt.EndTime := EncodeDate(2016,5,1) + d + 1 + random(5);
  plIt.Resource := Random(9);
  plIt.Text := 'Rental';
  plIt.Color := gcYellowgreen;
end;
end;

```



Introducing the TMS FNC RichEditor

With the TMS FNC Rich Editor you can assemble a WordPad-style editor or Outlook style mail application in a matter of minutes. TMS FNC Rich Editor comes with capabilities to do WYSIWYG editing of rich text with images, URLs, bullet lists, custom graphics, mail merging etc... To make development even faster, there is a pre-built toolbar for rich editor editing and formatting and non-visual components to facilitate the import and export from HTML & RTF files and that of course in all frameworks, operating systems and target devices supported.

In this introduction sample, drop the TTMSFNCRichEditor on the form as well as the TTMSFNCRichEditorFormatToolbar and assign the TTMSFNCRichEditor to TTMSFNCRichEditorFormatToolbar.RichEditor. Also add a TTMSFNCRichEditorHTMLIO and TTMSFNCRichEditorRTFIO non-visual component on the form and also assign the TTMSFNCRichEditor to TTMSFNCRichEditorHTMLIO.RichEditor and TTMSFNCRichEditorRTFIO.RichEditor.

The rich editor content can be initialized with following code to perform a mail-merge that uses here two merge fields: NAME and EMAIL.

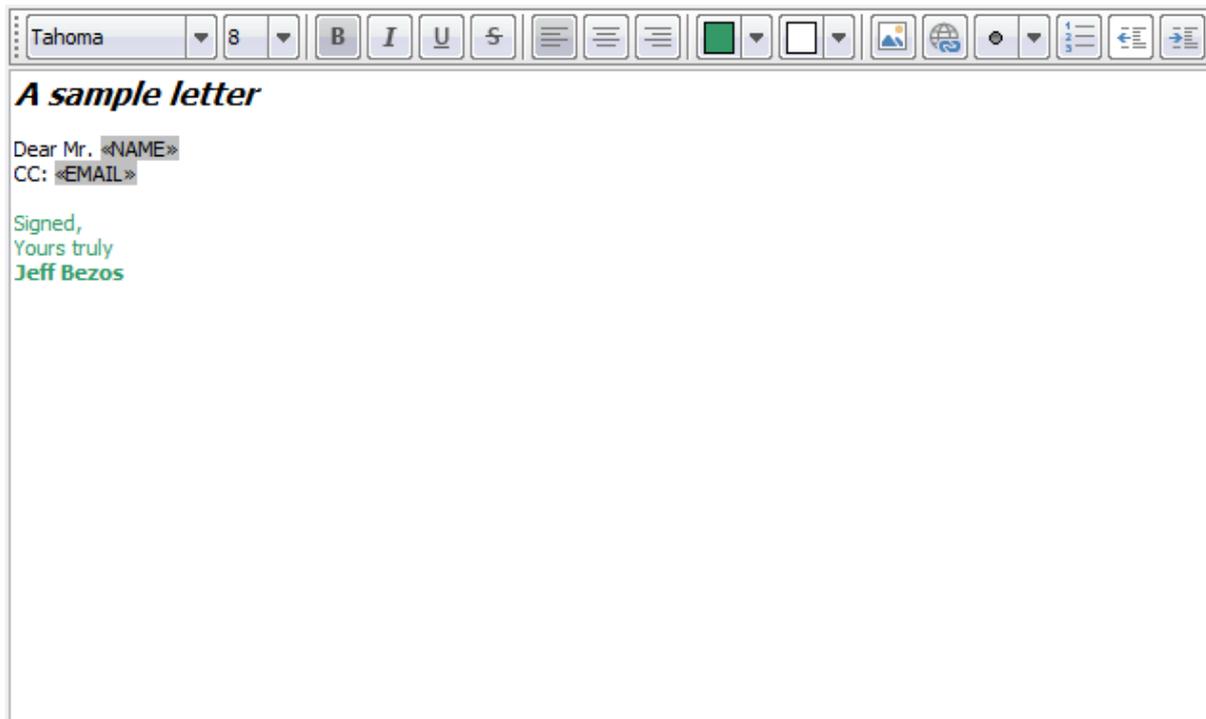
```

begin
  TMSFNCRichEditor1.AddText('Dear Mr. NAME');
  TMSFNCRichEditor1.AddLineBreak;
  TMSFNCRichEditor1.AddText('CC: EMAIL');
  TMSFNCRichEditor1.SelStart := 9;
  TMSFNCRichEditor1.SelLength := 4;
  TMSFNCRichEditor1.SetSelectionMergeField('NAME');

```

```
TMSFNCRichEditor1.SelStart := 21;
TMSFNCRichEditor1.SelLength := 5;
TMSFNCRichEditor1.SetSelectionMergeField('EMAIL');
end;
```

When the app is started, the text can be further decorated by editing & formatting via the toolbar. When it is ready, following code performs the merge with the NAME and EMAIL field and is exported to RTF via TTMSFNCRichEditorRTFIO and after this, the merge is undone:



```
var
  sl: TStringList;
begin
  sl := TStringList.Create;

  try
    sl.Add('NAME=Elon Musk');
    sl.Add('EMAIL=elon@tesla.com');
    TMSFNCRichEditor1.Merge(sl);
  finally
    sl.Free;
  end;

  TMSFNCRichEditorRTFIO1.Save('e:\tms\merge.rtf');
  TMSFNCRichEditor1.UnMerge;
end;
```

Introducing the TMS FNC TreeView

Finally, another large feature-packed control from the TMS FNC UI Controls set we want to introduce is the TMS FNC TreeView, TTMSFNCTreeView. This is a multi-column treeview control with regular mode and virtual mode and designed for and capable of using millions of nodes. In addition, the nodes support rich information, text attributes can be customized per node cell, HTML formatted text in node cells is possible, images, checkboxes can be added and optional inplace editing is available.

In this introduction we'd like to demonstrate the difference in regular (node collection) based mode and virtual mode when using the TTMSFNCTreeView. The first code snippet demonstrates an initialization of a 2 column treeview:

```
var
  tn,cn: TTMSFNCTreeViewNode;

begin
  TMSFNCTreeView1.BeginUpdate;

  TMSFNCTreeView1.Columns.Clear;
  TMSFNCTreeView1.Nodes.Clear;

  TMSFNCTreeView1.Columns.Add.Text := 'Country';
  TMSFNCTreeView1.Columns.Add.Text := 'Capital';

  tn := TMSFNCTreeView1.AddNode(nil);
  tn.Text[0] := 'Europe';

  cn := TMSFNCTreeView1.AddNode(tn);
  cn.Text[0] := 'Germany';
  cn.Text[1] := 'Berlin';

  cn := TMSFNCTreeView1.AddNode(tn);
  cn.Text[0] := 'France';
  cn.Text[1] := 'Paris';

  cn := TMSFNCTreeView1.AddNode(tn);
  cn.Text[0] := 'United Kingdom';
  cn.Text[1] := 'London';

  tn := TMSFNCTreeView1.AddNode(nil);
  tn.Text[0] := 'Asia';

  cn := TMSFNCTreeView1.AddNode(tn);
  cn.Text[0] := 'Japan';
  cn.Text[1] := 'Tokyo';

  cn := TMSFNCTreeView1.AddNode(tn);
  cn.Text[0] := 'China';
  cn.Text[1] := 'Peking';

  TMSFNCTreeView1.EndUpdate;
end;
```

Country	Capital
[-] Europe	
Germany	Berlin
France	Paris
United Kingdom	London
[-] Asia	
Japan	Tokyo
China	Peking

Important to note here is that the text in the multiple columns of the treeview can be simply accessed with an array indexed `Node.Text[]`: string property.

In a next step, we'll use the `TTMSFNCTreeView` in virtual mode and insert 1 million nodes! Columns are not virtual, so this must be initialized and to keep it simple, this will be initialized to one column:

```
begin
  TMSFNCTreeView1.Columns.Clear;
  TMSFNCTreeView1.Columns.Add;
  TMSFNCTreeView1.Columns[0].Text := 'Large treeview';
end;
```

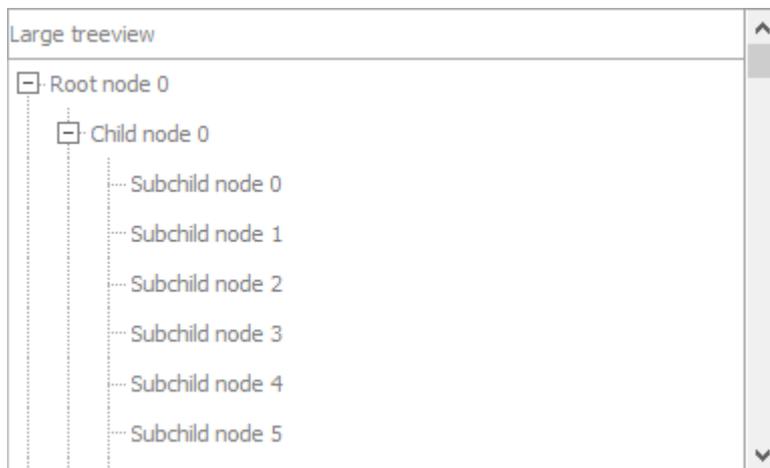
To use the `TTMSFNCTreeView` in virtual mode, two events are crucial: the `OnGetNumberOfNodes()` event and the `OnGetNodeText()` event. The first is triggered to know how many nodes at root level or child level should be added. The latter is used to retrieve the column text of the node. Let's start with the `OnGetNumberOfNodes` event. This event has parameters `ANode` and a var parameter `ANumberOfNodes`. `ANode` is either a node with `ANode.Level` set to -1 indicating the number of root level nodes is requested or it contains the node for which the number of child nodes is requested. With the `ANode.Level` property, you can know how many hierarchical levels deep the node is. In this example, we'll insert 1 million (100x100x100) nodes by inserting 100 root level nodes that have each 100 childs and each child has again 100 subchilds.

This event takes care of this:

```
procedure TForm1.TMSFNCTreeView1GetNumberOfNodes(Sender: TObject;
  ANode: TTMSFNCTreeViewVirtualNode; var ANumberOfNodes: Integer);
begin
  if ANode.Level = -1 then
    ANumberOfNodes := 100
  else
    if ANode.Level = 0 then
      ANumberOfNodes := 100
    else
      if ANode.Level = 1 then
        ANumberOfNodes := 100;
end;
```

Then, the other event for virtual node handling, `OnGetNodeText` is used to return the text for node columns. Note that this event will be triggered for each node and for each column for this node. The column for which the event is triggered is indicated with the `AColumn` parameter. As we have only 1 column in this example, this is ignored and the node text is directly returned:

```
procedure TForm1.TMSFNCTreeView1GetNodeText(Sender: TObject;
  ANode: TTMSFNCTreeViewVirtualNode; AColumn: Integer;
  AMode: TTMSFNCTreeViewNodeTextMode; var AText: string);
begin
  if ANode.Level = 0 then
    AText := 'Root node '+inttostr(ANode.Index)
  else
    if ANode.Level = 1 then
      AText := 'Child node '+inttostr(ANode.Index)
    else
      if ANode.Level = 2 then
        AText := 'Subchild node '+inttostr(ANode.Index);
      end;
end;
```



Conclusion

We hope this brief introduction of the major controls in the TMS FNC UI Pack whetted your appetite to start exploring the components, discovering the benefits and efficiency of having one UI component set to cover all the target operating systems you want to target and perhaps cook up your first Linux GUI apps with LCL. You can get the trial version for Delphi from <http://www.tmssoftware.com/site/tmsfnucipack.asp> and there is also a sample TV Guide project that can be used from VCL, FMX and LCL that you can obtain from <http://www.tmssoftware.com/site/blog.asp?post=335>

We're eager to learn how your experience is going and to hear your feedback, comments and further wishes and needs in this direction.