Getting started with TMS iCL

Introduction

TMS iCL is a component library. It stands for iOS Component Library. It is a Delphi component library and as such, it is accessible as Delphi objects with properties, methods, events. The components are in fact wrappers around the iOS operating system level defined controls, for now mostly visual controls. In iOS Objective C terminology, UIView types. In this respect it is very similar to the Delphi VCL standard components like TEdit, TButton, TListbox ... etc. These VCL controls are tiny wrappers around the Windows operating system controls like EDIT, BUTTON, LISTBOX ... These standard Delphi controls do not do much more than present the Windows user interface controls as Delphi classes with properties, method and events. In the case of TMS iCL, these controls are usable from a FireMonkey form, just like VCL controls are usable from a standard VCL form.

iCL vs FireMonkey

To be usable in iOS applications created with Delphi, iCL controls live on FireMonkey forms in a FireMonkey mobile application. An iCL control can be used on a FireMonkey form in the same way as a FireMonkey control is used. Internally, an iCL control is completely different from a FireMonkey control. Where a FireMonkey control is actually made up of a hierarchy of FireMonkey style objects (rectangles, labels, lines, scrollbar, animations) that are rendered on a canvas to emulate an operating system user interface control or to offer a new custom control, the iCL just wraps up an

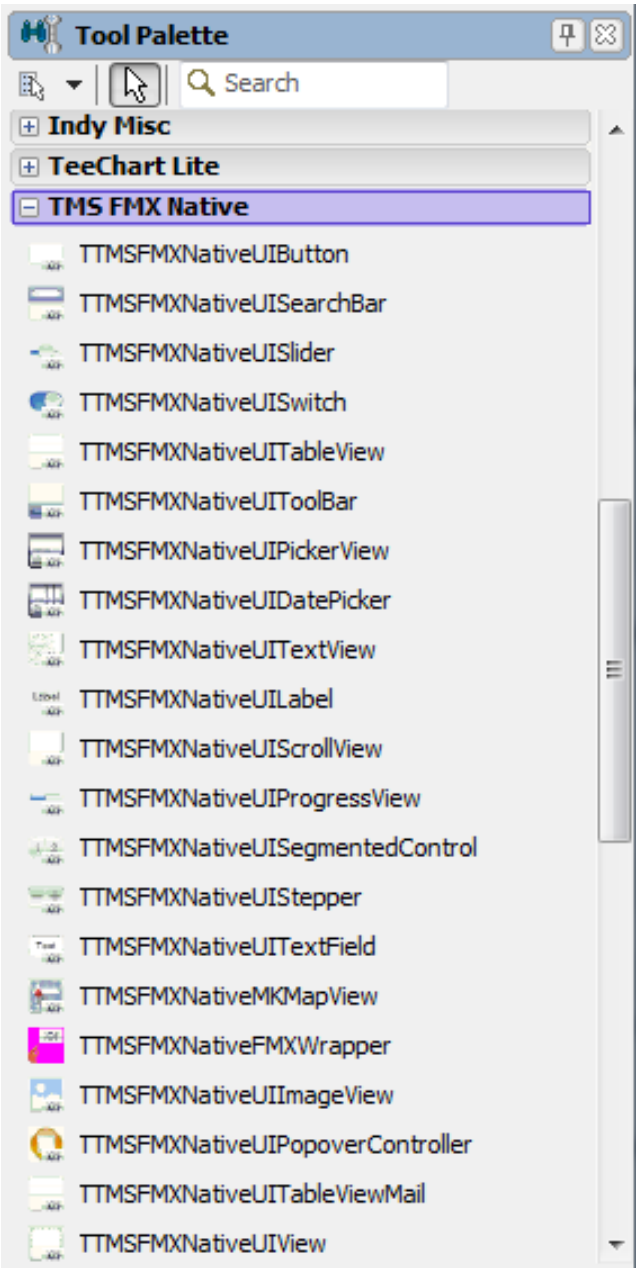
existing operating system control as Delphi object. This implies that iCL controls are not style-able, at least not in the same way a FireMonkey control is style-able. Fortunately, iCL controls and FireMonkey controls can coexist on the same FireMonkey form. This means you can mix & match controls and select what fits best for your application.

Advantages & disadvantages of iCL controls

The main advantages of iCL controls are: consistency and performance. Consistency is on various levels: the appearance of the iCL is 100% consistent with the native operating system user interface control as it is actually the operating system control itself. The appearance is also 100% consistent across different operating system versions. This means that an iCL switch control for example will have the look of iOS6 on a device running iOS 6 and the look of iOS7 on a device running iOS 7 without needing to do anything specific in the application. Not only will the appearance be 100% consistent but also the behavior. Response to touch, inertia in touch handling, physics like scroll bounces are guaranteed to behave 100% consistent with the operating system. The second advantage is performance. As the controls have been developed by Apple and part of the operating system, we can expect them to be fine-tuned for performance.
One might wonder, are there also disadvantages to iCL controls. Yes, a first disadvantage is that the controls can by definition not live in the design-time space. The Delphi IDE form designer runs only on Windows where the iOS control is absent and can thus only made accessible by emulating the control on the designer. For now, we do this by very basic emulation (to keep runtime code overhead very low). A second disadvantage is that the iCL will only work on iOS devices. As such, it won't be possible to create a cross platform application for Windows, Mac OS-X and iOS. And finally, an iCL control is not style-able like a FireMonkey control is. Such, it is less customizable.

Hands-on iCL

Let's get started using iCL controls in FireMonkey applications. After install of the controls, the new iCL controls are available in the Delphi XE4 component palette from FireMonkey iOS applications.



This is the list of controls that is currently available:

TMSFMXNativeUIButton: iOS button control that can have different button styles

TMSFMXNativeUISearchBar: iOS search entry

TMSFMXNativeUISlider: iOS slider control

TMSFMXNativeUISwitch: iOS on/off switch control

TMSFMXNativeUITableView: iOS tableview, high performance list with sections that can be set in group mode or plain mode

TMSFMXNativeUIToolBar: iOS toolbar control with option to add many predefined iOS toolbar buttons

TMSFMXNativeUIPickerView: iOS scrolling item picker with support for multiple columns

TMSFMXNativeUIDatePicker: iOS date or time picker or countdown control

TMSFMXNativeUITextView: iOS memo control

TMSFMXNativeUILabel: iOS label control

TMSFMXNativeUIScrollView: iOS scrolling area

TMSFMXNativeUIProgressView: iOS progressbar

TMSFMXNativeUISegmentedControl: iOS segment control

TMSFMXNativeUIStepper: iOS up/down step control

TMSFMXNativeUITextField: iOS edit control

TMSFMXNativeMKMapView: iOS map (Google maps on iOS 5, Apple maps on iOS6 or later)

TMSFMXNativeFMXWrapper: Control that wraps a FireMonkey form for use as view inside a native iOS control

TMSFMXNativeUIImageView: iOS image control (can display GIF,JPEG,TIFF,BMP,PNG,DIB,ICO,CUR,XBM images)

TMSFMXNativeUIPopoverController: iOS control to show other views as popup

TMSFMXNativeUIView: iOS base class view

Basic controls

The use of basic controls such as TMSFMXNativeUIButton, TMSFMXNativeUILabel, TMSFMXNativeUISwitch, etc… is really simple & straightforward. Drop the components on the form and you're ready to start using them in the same way you have used VCL controls. For example, the TMSFMXNativeUIButton has an OnClick event from where you handle button clicks. The TMSFMXNativeUILabel has a Text property via which you can configure the label text etc…

The TTMSFMXUITableView

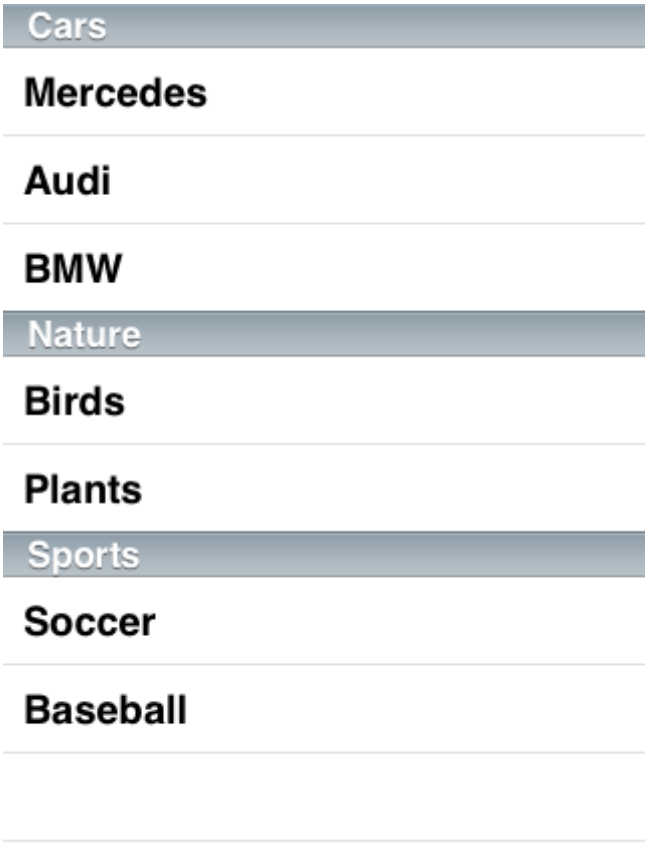
A more complex control is the TMSFMXNativeUITableView. A TTMSFMXUITableView consists of sections with items. At least one section should be added to the TTMSFMXUITableView. The TTMSFMXUITableView style is either plain (just long list of items) or grouped (where each section is displayed as a group of items)

At Delphi class level, the TTMSFMXUITableView sections and items are exposed as collections. The TMSFMXNativeUITableView has a collection of sections where each section has a header (string) and items collection. The TTMSFMXNativeUITableViewItem in the Items collection has properties Text: string to set the main text, a Description: string property to set a description text, a Bitmap: TBitmap to set an optional image associated with the item. At item level, it can also be controlled whether the item can be edited, deleted, moved,…

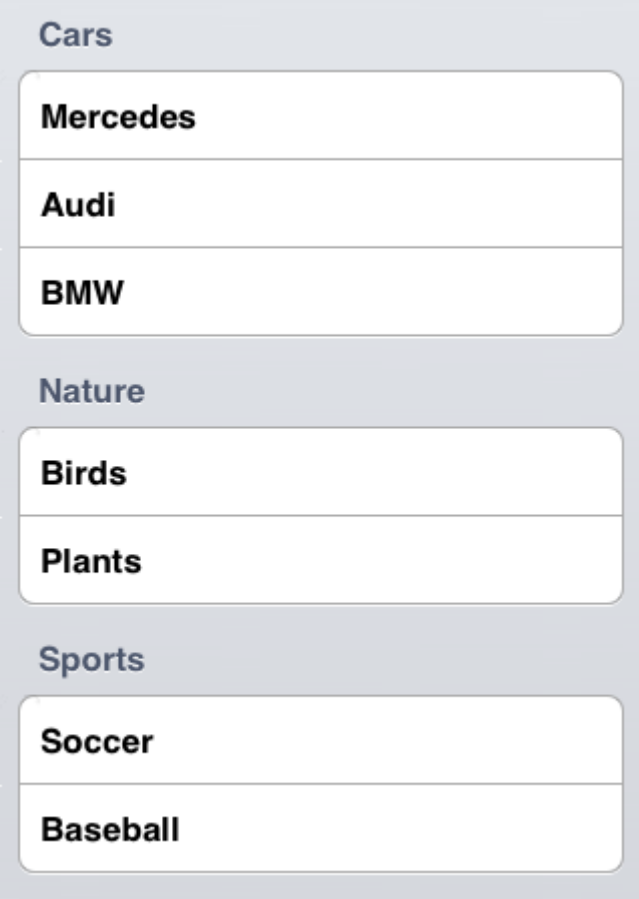
Adding items programmatically to the TTMSFMXUITableView can be done with:

```
var
  s: TTMSFMXNativeUITableViewSection;
begin
  s := TMSFMXNativeUITableView1.Sections.Add;
  s.Header := 'Cars';
  s.Items.Add.Text := 'Mercedes';
  s.Items.Add.Text := 'Audi';
  s.Items.Add.Text := 'BMW';
  s := TMSFMXNativeUITableView1.Sections.Add;
  s.Header := 'Nature';
  s.Items.Add.Text := 'Birds';
  s.Items.Add.Text := 'Plants';
  s := TMSFMXNativeUITableView1.Sections.Add;
  s.Header := 'Sports';
  s.Items.Add.Text := 'Soccer';
  s.Items.Add.Text := 'Baseball';
end;
```

Executing this code for a TMSFMXNativeUITableView that is in plain mode (TMSFMXNativeUITableView.Options.Layout = tvlNormal) results in:



Executing this code for a TMSFMXNativeUITableView that is in grouped mode (TMSFMXNativeUITableView.Options.Layout = tvlGrouped) results in:



When an item is clicked, the event OnItemSelect is triggered. This event returns the index of the section in which this item is and its index in the section's item collection. When we enable editing on the TTMSFMXUITableView, an Edit button appears in the top right corner. Editing is enabled via: TMSFMXNativeUITableView1.Options.Editing.Enabled: Boolean.

When in editing mode an item is deleted or moved, the events OnItemDelete or OnItemMove are triggered respectively.

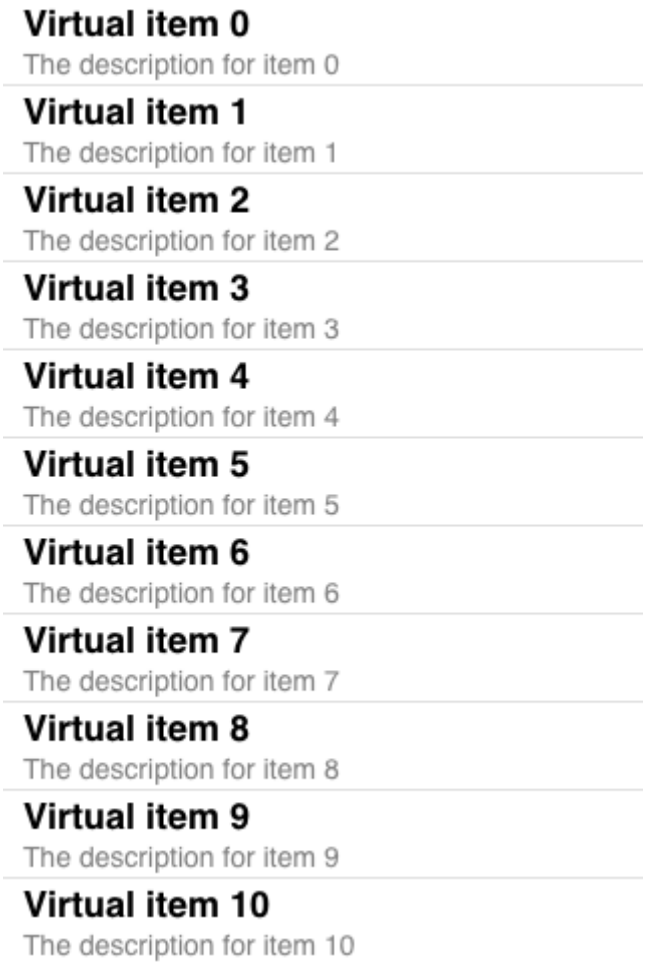
While we have exposed sections and items of the TTMSFMXUITableView as collection properties, the TTMSFMXUITableView component can also be easily used in virtual mode. In virtual mode, all data is not loaded at once, but via events, data is requested for the visible sections and items. For virtual mode, the first thing to do is to implement the events OnGetNumberOfSections and OnGetNumberOfRowsInSection. Via these events the number of sections in the TTMSFMXUITableView should be returned as well as the number of rows in each section. For a simple case of one section with 10.000 rows, this results in the code:

```
procedure TForm4.TMSFMXNativeUITableView1GetNumberOfRowsInSection(
  Sender: TObject; ASection: Integer; var ANumberOfRows: Integer);
begin
  if ASection = 0 then
    ANumberOfRows := 10000;
end;

procedure TForm4.TMSFMXNativeUITableView1GetNumberOfSections(Sender:
TObject;
```

```
  var ANumberOfSections: Integer);
begin
  ANumberOfSections := 1;
end;
```

Then we also need to return via events the text for items in the TTMSFMXUITableView via the event OnGetItemText and optionally also the description for the item via OnGetItemDescription.

```
procedure TForm4.TMSFMXNativeUITableView1GetItemText(Sender: TObject;
ASection,
  ARow: Integer; var AText: string);
begin
  if (ASection = 0) then
  begin
    AText := 'Virtual item ' + IntToStr(ARow);
  end;
end;

procedure TForm4.TMSFMXNativeUITableView1GetItemDescription(Sender:
TObject;
  ASection, ARow: Integer; var ADescription: string);
begin
  if (ASection = 0) then
  begin
    ADescription := 'The description for item ' + IntToStr(ARow);
  end;
end;
```



There are a lot more features and options in the TTMSFMXUITableView but that is beyond the scope of an iCL introduction article. The TTMSFMXUITableView has features to show a search or filter bar, a lookupbar, perform sorting, add a toolbar, having custom layouts in items, show detail views from item clicks and much more. We could dedicate an entire article just to the TTMSFMXUITableView alone.

The TTMSFMXMkMapView

A TMSFMXNativeMKMapView control provides an embeddable map interface, similar to the one provided by the iOS Maps application. You use this class as-is to display map information and to manipulate the map contents from your application. You can center the map on a given coordinate, specify the size of the area you want to display, and annotate the map with custom information.

Most important to get started with the TTMSFMXMkMapView is to set the region that is displayed in the map. This can be done with the SetRegion() call. This is method has 3 parameters: The top left longitude & latitude and the bottom right longitude & latitude as well as an animation parameter to decide whether the map is shown with or without animation on the specified coordinates. At the same time, when the user pans or zooms in de map and as a result of this the region changes, the event OnRegionDidChangeAnimated is triggered.

The second most important feature is to deal with annotations on the map. The annotations can either be a pin or a custom view / image. To add an annotation at the position where the user clicked, use the code:

```
var
  loc: TTMSFMXNativeMKMapLocation;
begin
  loc := TMSFMXNativeMKMapView1.XYToCoordinate(
    TMSFMXNativeMKMapView1.Width / 2,
    TMSFMXNativeMKMapView1.Height / 2);

  TMSFMXNativeMKMapView1.AddAnnotation(loc, 'Hello World', 'Subtitle');
end;
```

This code snippet adds an annotation in the center of the map with an image attached to it:

```
var
  loc: TTMSFMXNativeMKMapLocation;
  ann: TTMSFMXNativeMKAnnotation;
begin
  loc := TMSFMXNativeMKMapView1.XYToCoordinate(
    TMSFMXNativeMKMapView1.Width / 2,
    TMSFMXNativeMKMapView1.Height / 2);
  ann := TMSFMXNativeMKMapView1.AddAnnotation(loc, 'Hello World',
'Subtitle');
  ann.Bitmap.LoadFromFile(ExtractFilePath(ParamStr(0))+'pin.png');
end;
```

Conclusion

With TMS iCL we have tried to offer a comprehensive set of easy to use controls that give the perfect native look & feel and performance of iOS controls in Delphi FireMonkey applications. We have tried to expose as much as possible of the features that are in the iOS controls. In future version of TMS iCL we'll introduce more iOS controls and try to expose the maximum number of built-in capabilities in these iOS controls. TMS iCL offers as such an extra choice and flexibility to create FireMonkey applications with Delphi XE4 with the best possible user-interface you want to offer to your customers. A fully functional trial download of TMS iCL with PDF developers guide and sample applications can be downloaded via http://www.tmssoftware.com/site/tmsicl.asp