TMS iCL DEVELOPERS GUIDE

July 2013

Web: http://www.tmssoftware.com
Email: info@tmssoftware.com

Index

Availability

TMS iCL is a set of components for true native iOS application development and is available for Embarcadero Delphi XE4.

Online references

TMS software website:
http://www.tmssoftware.com

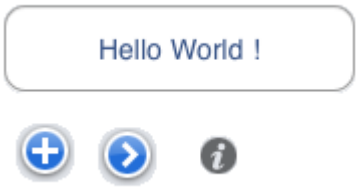
TMS iCL page:
http://www.tmssoftware.com/site/tmsiCL.asp

iOS Developer Library:
http://developer.apple.com/library/ios/navigation/

List of included components

- TMSFMXNativeUIButton
- TMSFMXNativeUISearchBar
- TMSFMXNativeUISlider
- TMSFMXNativeUISwitch
- TMSFMXNativeUITableView
- TMSFMXNativeUIToolBar
- TMSFMXNativeUIPickerView
- TMSFMXNativeUIDatePicker
- TMSFMXNativeUITextView
- TMSFMXNativeUILabel
- TMSFMXNativeUIScrollView
- TMSFMXNativeUIProgressView
- TMSFMXNativeUISegmentedControl
- TMSFMXNativeUIStepper
- TMSFMXNativeUITextField
- TMSFMXNativeMKMapView
- TMSFMXNativeFMXWrapper
- TMSFMXNativeUIImageView
- TMSFMXNativeUIPopoverController
- TMSFMXNativeUIView
- TMSFMXNativeUIImagePickerController
- TMSFMXNativeUITabBarController

TMSFMXNativeUIButton



Usage

An instance of TMSFMXNativeUIButton shows a native iOS Button on the screen.

Published Properties

Action	Property to assign an action combined with an action list.
Alignment	The technique to use for aligning the text.
Bitmap	Property used to show a bitmap on the Button.
Color	The background color of the Button.
Enabled	Enables or disables interaction with the Button.
LineBreakMode	The technique to use for wrapping and truncating the Button's text.
Style	The style of the Button. The Button style can be set to one of the following values: bsButtonTypeCustom bsButtonTypeRoundedRect (**Default**) bsButtonTypeDetailDisclosure bsButtonTypeInfoLight bsButtonTypeInfoDark bsButtonTypeContactAdd
TextColor	The color of the text of the Button.
TintColor	The color of the tint of the Button.

Visible	Shows or hides the Button.

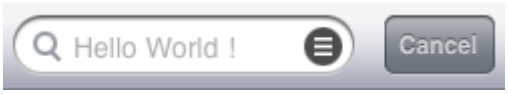
Public Properties

Button	Returns a reference to the native iOS UIButton.

Events

OnClick	Event called when clicking on the Button.

TMSFMXNativeUISearchBar



Usage

The TMSFMXNativeUISearchbar implements a text field control for text-based searches. The control provides a text field for entering text, a search button, a bookmark button, and a cancel button. The SearchBar does not actually perform any searches. Events can be used to implement the actions when text is entered and buttons are clicked.

Published Properties

PlaceHolder	The string that is displayed when there is no other text in the text field.
Prompt	A single line of text displayed at the top of the search bar.
SearchResultsButtonSelected	A Boolean value indicating whether the search results button is selected or not.
ShowsBookMarkButton	A Boolean value indicating whether the bookmark button is displayed or not.
ShowsCancelButton	A Boolean value indicating whether the cancel button is displayed or not.
ShowsSearchResultsButton	A Boolean value indicating whether the search results button is displayed or not.
Style	The style that specifies the toolbar appearance.
TintColor	The color used to tint the toolbar.
Text	The current or starting search text.
Translucent	Specifies whether the toolbar is translucent or not.
Visible	Shows or hides the SearchBar.

Public Properties

SearchBar	Returns a reference to the native iOS

	UISearchBar.

Events

OnBookmarkButtonClicked	Event called when the bookmark button is clicked.
OnCancelButtonClicked	Event called when the cancel button is clicked.
OnResultsListButtonClicked	Event called when the results list button is clicked.
OnSearchButtonClicked	Event called when the search button is clicked.
OnSelectedScopeButtonIndexDidChange	Event called when the selected scope changed.
OnShouldBeginEditing	Event called when the editing should begin.
OnShouldChangeTextInRange	Event called when the text in range should be changed.
OnShouldEndEditing	Event called when the editing should end.
OnTextDidBeginEditing	Event called when the editing of the text did begin.
OnTextDidChange	Event called when the text did change.
OnTextDidEndEditing	Event called when the editing of the text did end.

TMSFMXNativeUISlider

Usage

A TMSFMXNativeUISlider object is a visual control used to select a single value from a continuous range of values. Sliders are always displayed as horizontal bars. An indicator, or thumb, notes the current value of the Slider and can be moved by the user to change the setting.

Published Properties

MaximumValue	Contains the maximum value of the Slider.
MinimumValue	Contains the minimum value of the Slider.
Value	Contains the Slider's current value.
Visible	Shows / hides the Slider.

Public Properties

Slider	Returns a reference to the native iOS UISlider.

Events

OnTouchDown	Event called when the Slider's touch down is performed.
OnTouchUpInside	Event called when the Slider's touch up inside performed
OnTouchUpOutside	Event called when the Slider's touch up outside is performed.
OnValueChanged	Event called when the Slider's value has changed.

TMSFMXNativeUISwitch

Usage

The TMSFMXNativeUISwitch class is typically used to create and manage On/Off Buttons.

Properties

Value	A Boolean value that determines the off/on state of the Switch.
Visible	Shows / hides the Switch.

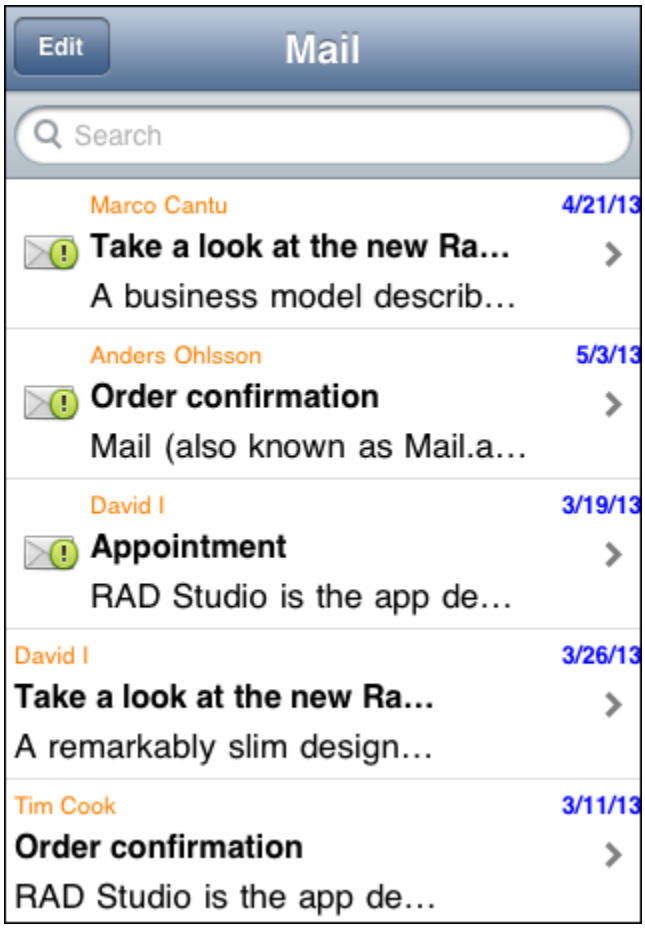
Methods

Switch	Returns a reference to the native iOS UISwitch.

Events

OnValueChanged	Event called when the off/on state of the Switch changes.

TMSFMXNativeUITableView



Usage

An instance of TMSFMXNativeUITableView is a means for displaying and editing hierarchical lists of information.

A table view is made up of at least one section, each with its own items. Sections are identified by their index number within the table view, and items are identified by their index number within a section. Any section can optionally be preceded by a section header.

Published Properties

DetailView	DetailView of the TableView used to display content from another TMS FMX Native UI Control linked to the DetailView of an item within a section.
MasterTableView	MasterTableView of the sub TableView when multiple TableView instances are linked to each

	other and represent a hierarchical master-detail structure.
Options	A set of configurable options for the TableView.
Options → AllowsMultipleSelection	A Boolean value that determines whether users can select more than one row outside of editing mode or not.
Options → AllowsMultipleSelectionDuringEditing	A Boolean value that controls whether users can select more than one item simultaneously in editing mode or not.
Options → AllowsSelection	A Boolean value that determines whether users can select a row or not.
Options → AllowsSelectionDuringEditing	A Boolean value that determines whether users can select items while the receiver is in editing mode or not.
Options → Editing	A set of configurable editing options for the TableView.
Options → Editing → EditButton	Shows an edit button in the toolbar which toggles the TableView from normal to edit mode or vice versa. (Options.Toolbar and Options.Editing.Enabled properties need to be true)
Options → Editing → Enabled	Enables or disables editing capabilities in the TableView.
Options → Header	The header of the TableView displayed in the toolbar.
Options → Layout	The Layout of the TableView. The TableView can have 2 layout modes: plain and grouped layout mode.
Options → LookUp	A set of configurable lookup options for the TableView.
Options → LookUp → Items	A collection of custom lookup items.
Options → LookUp → Mode	The lookup mode of the TableView. The mode can be set to alphabetic, alphanumeric, numeric or custom. In case of custom, the lookup bar in the TableView is filled with items from the Items collection under the Lookup property. The linked

	can be done by specifying an ID on the item and a LookUpID on the section.
Options → Refreshing	A set of configurable refreshing options for the TableView.
Options → Refreshing → AutoEnd	Automatically ends refresh operation after BeginRefreshing is called or a swipe down operation is performed.
Options → Refreshing → Enabled	Enables Refreshing, disabled by default.
Options → Refreshing → TintColor	Sets the tint-color of the Refresh indicator on the TableView.
Options → RowHeight	The default rowheight of the TableView items. The rowheight can be configured per item with the Height property.
Options → Scrolling	A set of configurable scrolling options for the TableView.
Options → Scrolling → AlwaysBounceHorizontal	A Boolean value that determines whether bouncing always occurs when horizontal scrolling reaches the end of the content view or not.
Options → Scrolling → AlwaysBounceVertical	A Boolean value that determines whether bouncing always occurs when vertical scrolling reaches the end of the content or not.
Options → Scrolling → Bounces	Bounces the view horizontally or vertically depending on the AlwaysBounceHorizontal and AlwaysBounceVertical properties.
Options → Scrolling →DirectionalLockEnabled	A Boolean value that determines whether scrolling is disabled in a particular direction or not.
Options → Scrolling →Enabled	A Boolean value that determines whether scrolling is enabled or not.
Options → Scrolling →ShowsHorizontalScrollIndicator	A Boolean value that controls whether the horizontal scroll indicator is visible or not.
Options → Scrolling →ShowsVerticalScrollIndicator	A Boolean value that controls whether the vertical scroll indicator is visible or not.
Options → Searching	A set of Configurable searching options for the

	TableView.
Options → Searching → Mode	Filtering or Searching mode. In Filtering mode, the items are listed that match the characters entered in the SearchBar. In Searching mode, the item that matches the characters entered in the SearchBar is being visualized after clicking on the search button.
Options → Searching → ScrollMode	Defines the way of visualizing the item after pressing the search button in search mode. There are 2 ways of visualizing the item: scroll to - or scroll and select the item.
Options → Searching → ScrollPosition	The position in the TableView (top, middle, bottom) to which a given row is scrolled when using searching mode.
Options → SeparatorColor	The color of separator items in the table view.
Options → SeparatorStyle	The style for items used as separators.
Options → ToolBar	Shows / hides the toolbar on the TableView.
Sections	A collection of Sections used in the TableView.
Sections[Index] → Header	The header of the section.
Sections[Index] → Footer	The footer of the section.
Sections[Index] → Items	Collection of items within a section
Sections[Index] → Items[Index] → AccessoryType	The type of standard accessory view the item should use (normal state). Standard accessory types are: atTableViewCellAccessoryNone atTableViewCellAccessoryDisclosureIndicator atTableViewCellAccessoryDetailDisclosureButton atTableViewCellAccessoryCheckmark
Sections[Index] → Items[Index] → AccessoryView	A view that is used, typically as a control, on the right side of the item (normal state).

Sections[Index] → Items[Index] → Bitmap	Sets the image of an item in the TableView.
Sections[Index] → Items[Index] → BitmapFile	A direct link to an image file located in the root or documents directory.
Sections[Index] → Items[Index] → BitmapLink	A link to another TBitmap instance which can be used multiple times to save resources.
Sections[Index] → Items[Index] → BitmapSize	The size of an image used to resize. The BitmapSize is -1 by default which will load the original size of the image The image is resized with aspect ratio.
Sections[Index] → Items[Index] → CanMove	Sets whether an item can be moved to another location in the TableView or not.
Sections[Index] → Items[Index] → Enabled	Enables / disables the item.
Sections[Index] → Items[Index] → Description	The description of the item.
Sections[Index] → Items[Index] → DetailView	A DetailView that is used when the user selects an item. The DetailView is pushed in the View linked to the TableView's DetailView property. The DetailView on item level and on TableView can be linked to other types of TMS FMX Native UI controls.
Sections[Index] → Items[Index] → EditingAccessoryType	The type of standard accessory view the item should use (editing state). Standard accessory types are: atTableViewCellAccessoryNone atTableViewCellAccessoryDisclosureIndicator atTableViewCellAccessoryDetailDisclosureButton atTableViewCellAccessoryCheckmark
Sections[Index] → Items[Index] → EditingAccessoryView	A view that is used, typically as a control, on the right side of the item (editing state).
Sections[Index] → Items[Index] → EditStyle	The style of the item when editing the TableView. The style can be set to delete or insert the item. When editing occurs, an item is respectively

	delete or inserted.
Sections[Index] → Items[Index] → Height	The height of an item in the TableView when a value different from -1 is specified. If the Value is -1 the RowHeight property on Options level is used.
Sections[Index] → Items[Index] → ShowEditMenu	Shows a "Copy" edit menu when a tap and hold operation occurs on an item.
Sections[Index] → Items[Index] → Style	The Style of an item. Various styles can be applied UITableViewCellStyleDefault A simple style for an item with a text label (black and left-aligned) and an optional image view. UITableViewCellStyleValue1 A style for an item with a label on the left side of the item with left-aligned and black text; on the right side is a label that has smaller blue text and is right-aligned. UITableViewCellStyleValue2 A style for an item with a label on the left side of the item with text that is right-aligned and blue. On the right side of the item is another label with smaller text that is left-aligned and black. UITableViewCellStyleSubtitle A style for an item with a left-aligned label across the top and a left-aligned label below it in smaller gray text.
Sections[Index] → Items[Index] → SubDetailView	A SubDetailView that is used when the user selects an item. The SubDetailView is pushed in place of the main TableView. The SubDetailView on item level can be linked to other types of TMS FMX Native UI controls.
Sections[Index] → Items[Index] → Text	The text of the item.

Sections[Index] → LookUpID	The LookUpID of the section that is used in combination with custom lookup items.
Visible	Shows / Hides the TableView.

Public Properties

EditButton	Returns a reference to the native iOS UIBarButton that is used in the ToolBar for editing purposes.
NavigationController	Returns a reference to the native iOS UINavigationController used to navigate in hierarchical structure when a SubDetailView item relation is setup.
SearchBar	Returns a reference to the native iOS UISearchBar.
SearchDisplayController	Returns a reference to the native iOS UISearchDisplayController that is used to display the search results.
TableView	Returns a reference to the native iOS UITableView.
TableViewController	Returns a reference to the native iOS UITableViewController.

Methods

BeginUpdate / EndUpdate	Wrapping code to block direct updates to the TableView. This is done for performance when loading a large amount of items and content.
BeginRefreshing / EndRefreshing	Shows a refreshing indicator on the TableView to show that the TableView is currently refreshing / updating its contents. Always combine the two methods to make sure that the indicator is hidden when the refresh operation is finished.
Edit	Method used to set the TableView in edit mode, if the editing is enabled in the options through Options.Editing.Enabled.

EditDone	Method used to finish editing mode and put the TableView back to normal mode.
GetItem(ASection, ARow: Integer; ASearchFilterList: Boolean = True): TTMSFMXNativeUITableViewItem;	Function that returns an item for the current section and row in the collection and optionally searches in the filtered list when needed.
HideDetailView	Method to return the TableView back to the master view when a master-detail hierarchy is setup.
IsEditing: Boolean	Function that returns a Boolean whether the TableView is in edit mode or not.
IsFiltering: Boolean	Function that returns a Boolean whether the TableView is in filtering mode or not.
UpdateSelectionAtRow(ASection, ARow: Integer)	Updates a row at a section
UpdateTableView	Update the complete TableView.

Events

OnAddItemToFilterList	Event called when an item is added to the filter list when filtering is enabled in the TableView and a text is entered in the SearchBar.
OnBeginRefreshing	Event called when refreshing begins, triggered when swiping down, or calling BeginRefreshing.
OnCanMoveItem	Event called to return a Boolean whether an item can be moved from and to a location or not.
OnEditEnd	Event called when editing ended.
OnEditStart	Event called when editing started.
OnEndRefreshing	Event called when refreshing ends, triggered when the refreshing operation is complete, or when calling EndRefreshing.
OnFilterItemsForText	Event called when filtering the TableView when a text is entered in the SearchBar.
OnGetItemAccessoryType	Event called to return an Accessory Type for an item in normal mode.

OnGetItemAccessoryView	Event called to return an Accessory View for an item in normal mode. The AccessoryView can be linked to another TMS FMX Native UI Control.
OnGetItemAppearance	Event called to customize text, description, background and selection colors and fonts.
OnGetItemBitmap	Event called to return a Bitmap for an item.
OnGetItemDescription	Event called to return a Description for an item.
OnGetItemDetailView	Event called to return a DetailView for an item. The DetailView can be linked to another TMS FMX Native UI Control.
OnGetItemEditingAccessoryType	Event called to return an Accessory Type for an item in edit mode.
OnGetItemEditingAccessoryView	Event called to return an Accessory View for an item in editing mode. The AccessoryView can be linked to another TMS FMX Native UI Control.
OnGetItemEditingStyle	Event called to return an editing style for an item.
OnGetItemFilterText	Event called that returns the filter text that is used to compare with the text entered in the SearchBar.
OnGetItemHeight	Event called that returns a height for an item.
OnGetItemStyle	Event called to return a style for an item.
OnGetItemSubDetailView	Event called to return a SubDetailView for an item. The SubDetailView can be linked to another TMS FMX Native UI Control.
OnGetItemText	Event called to return the text of an item.
OnGetNumberOfRowsInSection	Event called that specifies the number of rows in a section.
OnGetNumberOfSections	Event called that specifies the number of sections in a TableView.
OnGetSectionForSectionIndexTitle	Event called that returns the section for a specific index title. The section index title is an equivalent for the lookup characters in the lookup bar.

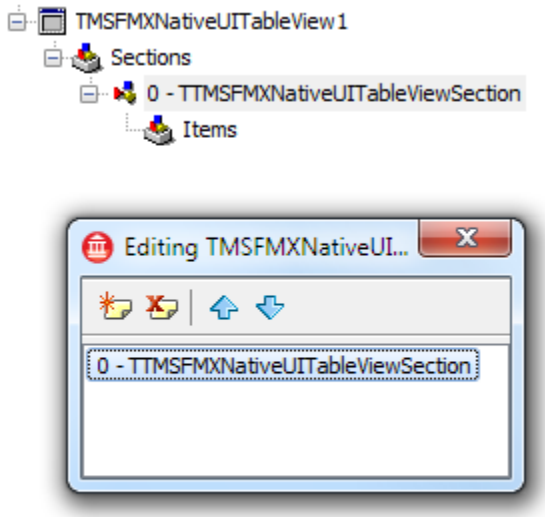
OnGetSectionIndexTitles	Event called that returns an array of section index titles. The section index title is an equivalent for the lookup characters in the lookup bar.
OnGetTitleForHeaderInSection	Event called that returns a header title for a section.
OnIsItemInFilterCondition	Event called to know if an item matches a specific filter condition.
OnItemAccessoryButtonClick	Event called when clicking on the Accessory Button when the AccessoryType has been set to atTableViewCellAccessoryDetailDisclosureButton.
OnItemBeforeShowDetailView	Event called before navigating from the master to the detail when a master-detail hierarchy is setup.
OnItemCompare	Event called when comparing 2 items for sorting capabilities. Through this event, custom sorting can be applied.
OnItemDelete	Event called when an item will be deleted.
OnItemDeleted	Event called when an item is deleted.
OnItemDeSelect	Event called when an item is deselected.
OnItemInsert	Event called when an item will be inserted.
OnItemInserted	Event called when an item is inserted.
OnItemMove	Event called when an item will be moved.
OnItemMoved	Event called when an item is moved.
OnItemSelect	Event called when an item is selected.
OnSearchEnd	Event called when searching has ended.
OnSearchStart	Event called when searching has started.
OnShouldShowEditMenuForItem	Event called that returns a Boolean whether a Copy edit menu should be shown for an item or not.

Public Events

OnItemCustomizeCell	Event used to customize a cell after all properties are applied.
OnItemCreateCell	Event called when creating a cell. This event can be used to add additional native UI controls and can be combined with the OnItemCustomizeCell to apply content.
OnItemPerformCopyAction	Event called when the Copy action is clicked after the copy menu has been shown by tap-holding on the item.
OnTableViewLoadMore	Event called when the tableview reaches the end. This event can be used to load more items when scrolling.

Adding Sections and Items

The TableView consists of (optionally multiple) sections and items. To add a section at designtime, click on the TableView, select the sections collection and click on the add button:



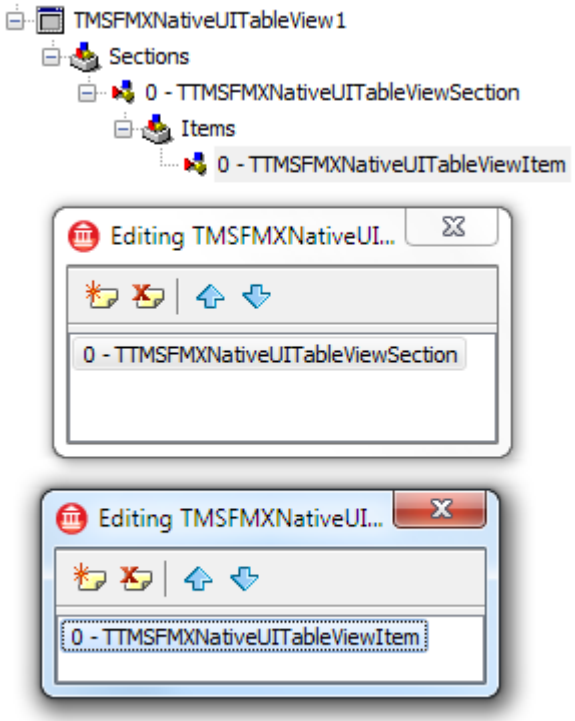
Each section has a Header property that is empty by default. To visualize sections in the TableView, enter a value in this property such as “Cars”, “Nature”or “Sport”, … .

Sections can also be added programmatically:

```
var
  s: TTMSFMXNativeUITableViewSection;
```

```
begin
  s := TMSFMXNativeUITableView1.Sections.Add;
  s.Header := 'Cars';

  s := TMSFMXNativeUITableView1.Sections.Add;
  s.Header := 'Nature';

  s := TMSFMXNativeUITableView1.Sections.Add;
  s.Header := 'Sports';
```

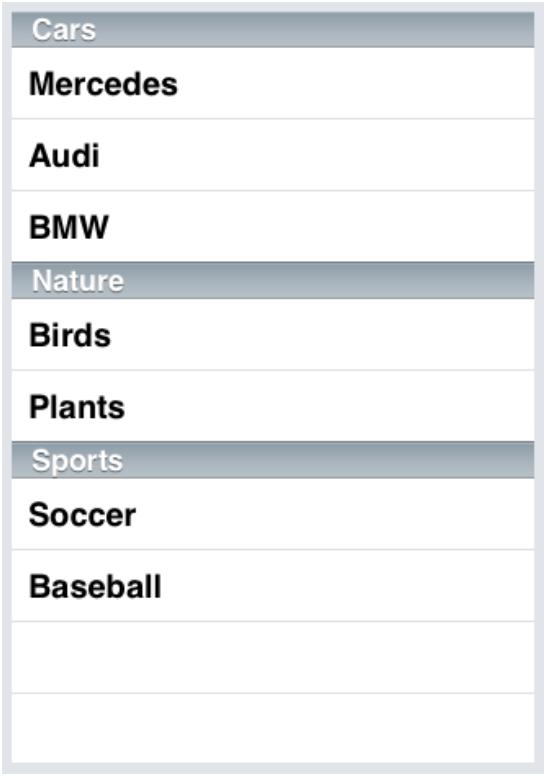
Each section has a collection of items. To add an item to a section at designtime, click on the previously created section and double-click on the items collection. In the editor, click on the add button to add an item.



An item can also be added programmatically. If we take the previous snippet that creates sections, we can add items to those sections:

```
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';
```

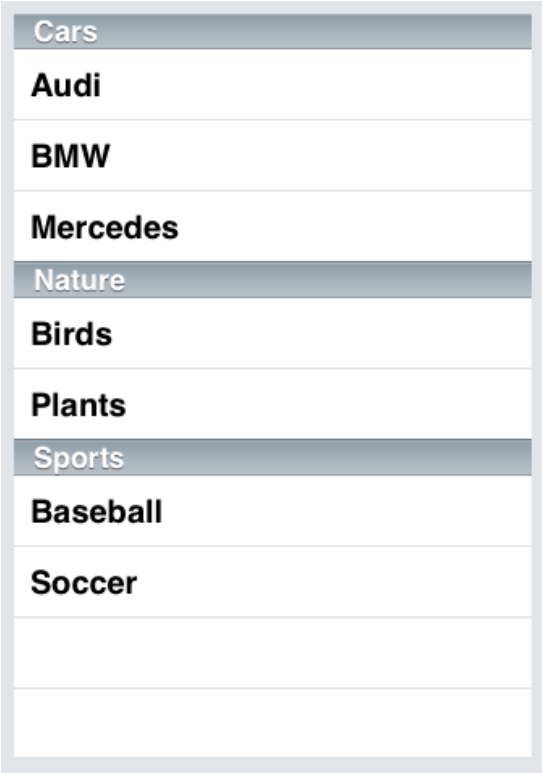


Sorting

The TableView also supports built-in sorting. Sorting can be applied to sections and items. Call the procedure Sort on the section or items collection. Optional parameters can be passed for an ascending or descending order.

If we take the sample and apply sorting, the items will be sorted per section:

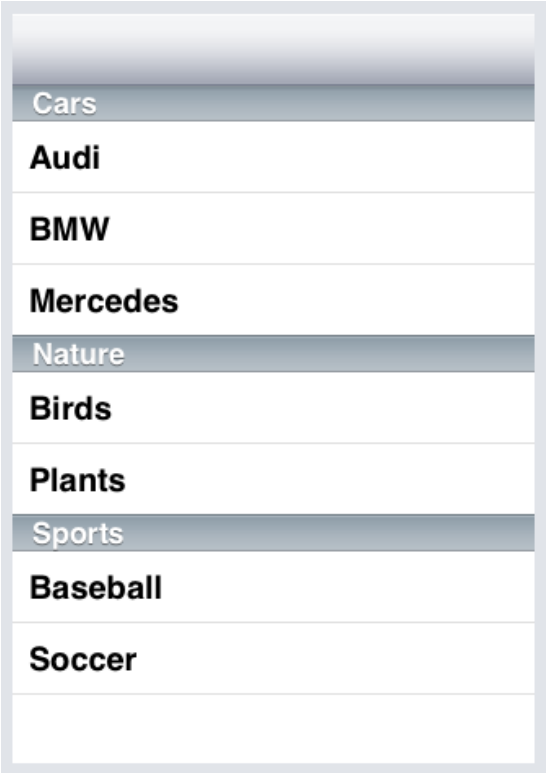
```
var
  s: TTMSFMXNativeUITableViewSection;
begin
  s := TMSFMXNativeUITableView1.Sections.Add;
  s.Header := 'Cars';
  s.Items.Add.Text := 'Mercedes';
  s.Items.Add.Text := 'Audi';
  s.Items.Add.Text := 'BMW';
  s.Items.Sort;

  s := TMSFMXNativeUITableView1.Sections.Add;
  s.Header := 'Nature';
  s.Items.Add.Text := 'Birds';
  s.Items.Add.Text := 'Plants';
  s.Items.Sort;

  s := TMSFMXNativeUITableView1.Sections.Add;
  s.Header := 'Sports';
  s.Items.Add.Text := 'Soccer';
  s.Items.Add.Text := 'Baseball';
  s.Items.Sort;
```



Toolbar

The TableView has built-in support for displaying a toolbar, that is used for Master-Detail navigation and/or editing. To display the toolbar, set Options.ToolBar to true.

```
TMSFMXNativeUITableView1.Options.ToolBar := True;
```

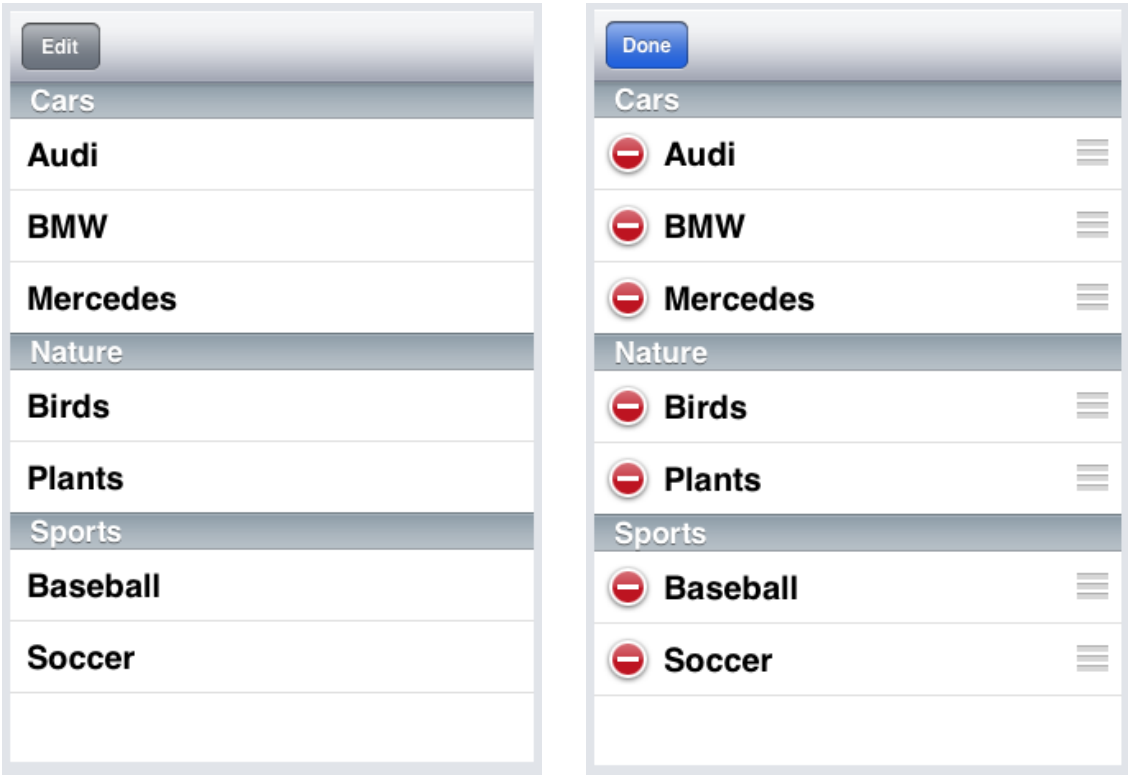


Editing

When the toolbar is enabled, an optional edit button can be displayed, that toggles the TableView between normal and edit mode. By default, the Options.Editing.EditButton property is true, but to enable editing, the Options.Editing.Enabled needs to be set to true. This gives the result below.

Clicking on the edit button sets the TableView in edit mode and modifies the button so the TableView can be reverted back to normal mode.

```
TMSFMXNativeUITableView1.Options.Editing.Enabled := True;
```



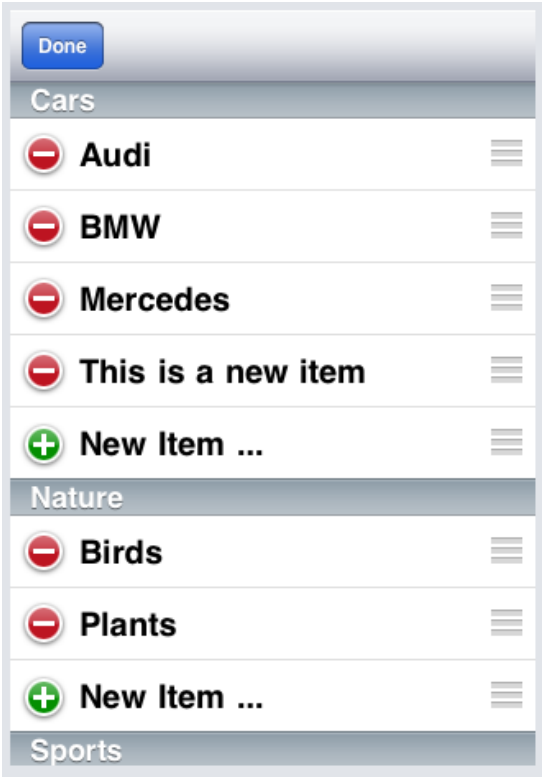
When clicking on the delete indicator next to the item, the item will be deleted from the collection. In normal modethere is also an ability to delete the item on a swipe gesture over the item. A Delete button appears that executes the same functionality as in editing mode.

Each item has a EditStyle property that is esTableViewCellEditingStyleDelete by default. The EditStyle can be set to esTableViewCellEditingStyleInsert to show a plus button, or set to esTableViewCellEditingStyleNone to disallow editing capabilities of an item.

Below is a sample code that adds an extra insertable item in the TableView. When the insert button is clicked, the OnItemInserted event is called which adds an additional item to the tableview. In this event, properties of the newly created item can be modified.

```
var
  s: TTMSFMXNativeUITableViewSection;
begin
  TMSFMXNativeUITableView1.Options.ToolBar := True;
  TMSFMXNativeUITableView1.Options.Editing.Enabled := True;

  s := TMSFMXNativeUITableView1.Sections.Add;
  s.Header := 'Cars';
  s.Items.Add.Text := 'Mercedes';
  s.Items.Add.Text := 'Audi';
  s.Items.Add.Text := 'BMW';
  s.Items.Sort;
```

```
  with s.Items.Add do
  begin
    Text := 'New Item ...';
    EditStyle := esTableViewCellEditingStyleInsert;
  end;

  s := TMSFMXNativeUITableView1.Sections.Add;
  s.Header := 'Nature';
  s.Items.Add.Text := 'Birds';
  s.Items.Add.Text := 'Plants';
  s.Items.Sort;
  with s.Items.Add do
  begin
    Text := 'New Item ...';
    EditStyle := esTableViewCellEditingStyleInsert;
  end;

  s := TMSFMXNativeUITableView1.Sections.Add;
  s.Header := 'Sports';
  s.Items.Add.Text := 'Soccer';
  s.Items.Add.Text := 'Baseball';
  s.Items.Sort;
  with s.Items.Add do
  begin
    Text := 'New Item ...';
    EditStyle := esTableViewCellEditingStyleInsert;
  end;
end;

procedure TForm1.TMSFMXNativeUITableView1ItemInserted(Sender:
TObject;
  ASection, ARow: Integer);
var
  it: TTMSFMXNativeUITableViewItem;
begin
  it := TMSFMXNativeUITableView1.Sections[ASection].Items[ARow];
  it.Text := 'This is a new item';
end;
```



On the left side, there is a move item indicator that allows moving item within sections or to another section. With the CanMove property, this can optionally be controlled per item.

Searching / Filtering

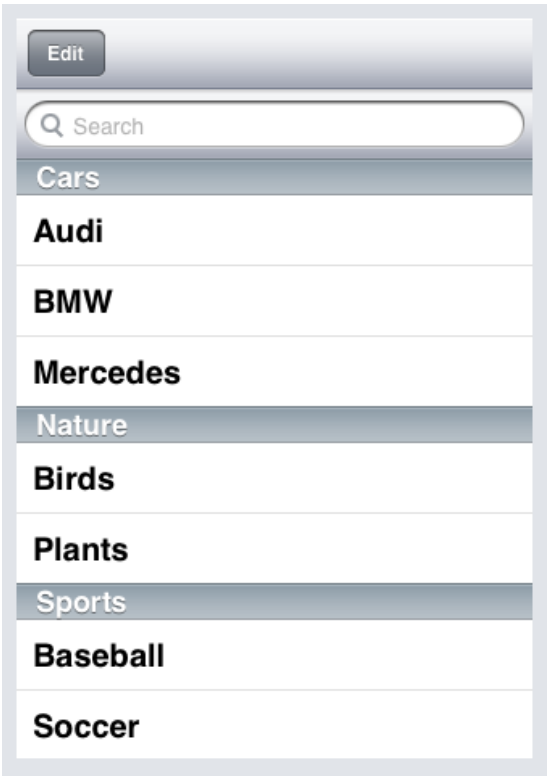
The TableView has built-in support for searching / filtering. With the Options.Searching.Mode property a SearchBar can be enabled to allow searching or filtering. With filtering, the items that are listed in the TableView are based on the text entered in the SearchBar. Only the TableView items that match the filter condition are listed. Filtering can be modified with events that control the filter condition and the results list.

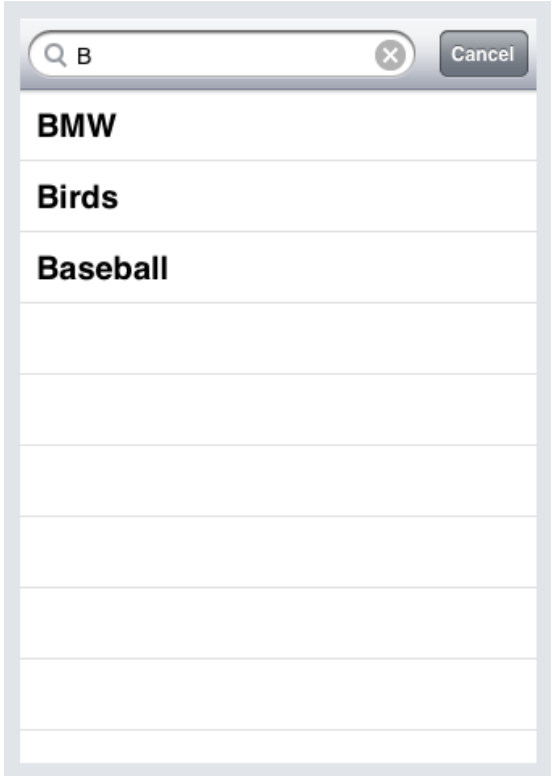
When enabling searching mode, the items remain listed but are scrolled to when the search condition is matched and the search button on the keyboard is pressed.

Below is a sample on how to enable filtering and a sample of a filter result.

```
var
  s: TTMSFMXNativeUITableViewSection;
begin
  TMSFMXNativeUITableView1.Options.ToolBar := True;
  TMSFMXNativeUITableView1.Options.Editing.Enabled := True;
  TMSFMXNativeUITableView1.Options.Searching.Mode := smFiltering;

  s := TMSFMXNativeUITableView1.Sections.Add;
```

```
s.Header := 'Cars';
s.Items.Add.Text := 'Mercedes';
s.Items.Add.Text := 'Audi';
s.Items.Add.Text := 'BMW';
s.Items.Sort;

s := TMSFMXNativeUITableView1.Sections.Add;
s.Header := 'Nature';
s.Items.Add.Text := 'Birds';
s.Items.Add.Text := 'Plants';
s.Items.Sort;

s := TMSFMXNativeUITableView1.Sections.Add;
s.Header := 'Sports';
s.Items.Add.Text := 'Soccer';
s.Items.Add.Text := 'Baseball';
s.Items.Sort;
```





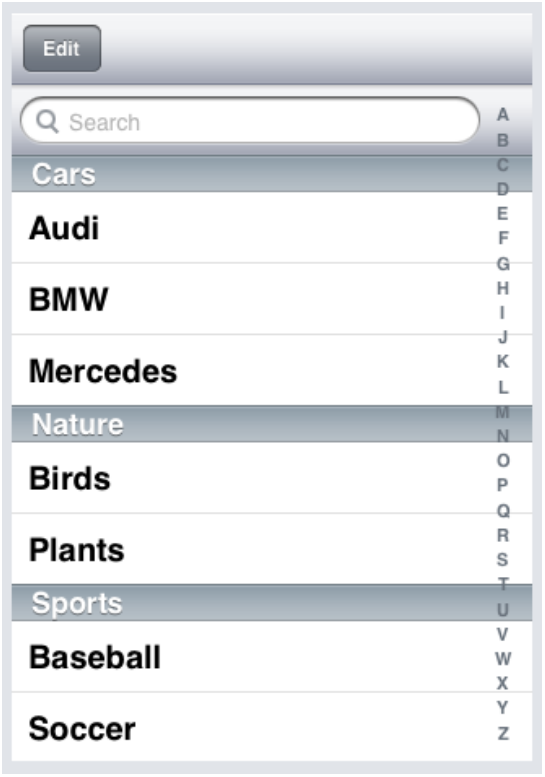
More information on events is explained in the events table overview.

Lookup

Lookup enables the ability to show a list of characters or indexes that are linked to the section header. When clicking or swiping over the lookup bar, the TableView scrolls to the correct section. This is particularly helpful if there are multiple sections and items that extend the height of the TableView.

To enable lookup, set Options.Lookup.Mode to lmAlphaBetic to enable an AlphaBetic list of lookup indexes in the TableView. The Mode can be changed to allow AlphaNumeric, Numeric or custom. When choosing the last options, the Options.Lookup.Items collection is used to fill up the lookup list with custom indexes that are linked to a section through the ID. Sections have a LookupID that needs to match one of the custom items in the Lookup.

Below is a sample screenshot that shows the lookup bar on the right of the TableView.

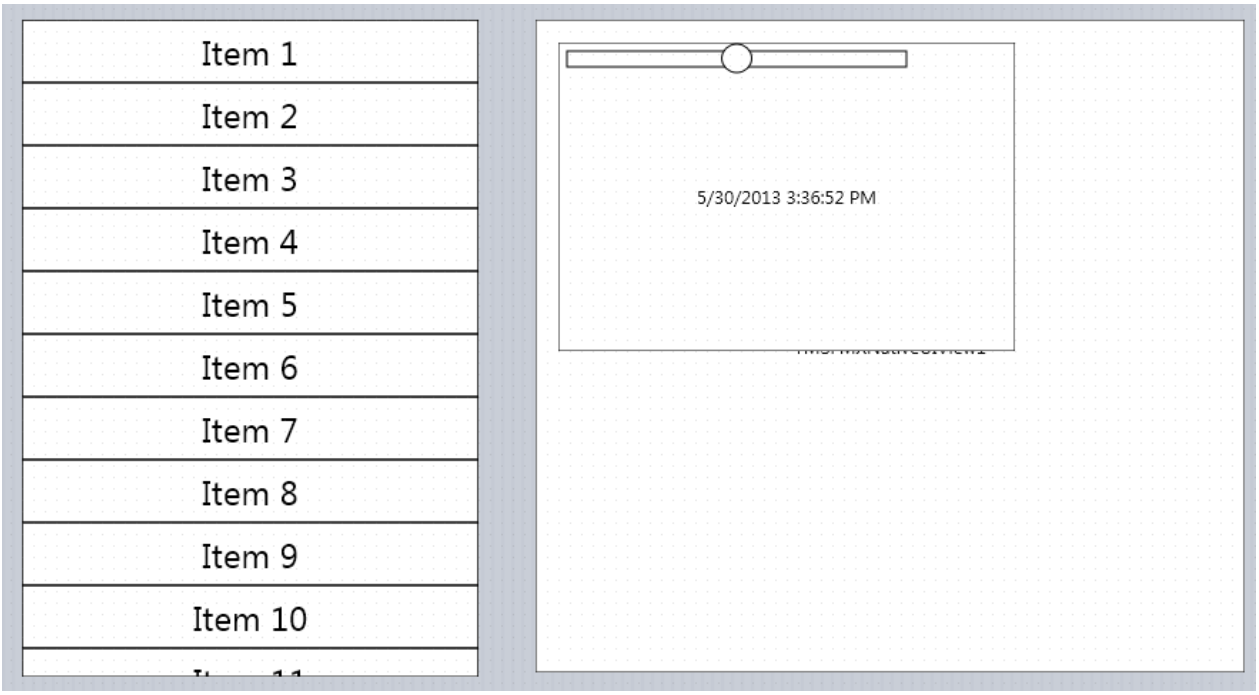


DetailView and SubDetailView

Each item in the TableView has a DetailView and SubDetailView property. With the DetailView property, another TMS FMX Native UI control can be linked and displayed when clicking on the item. On TableView level there is also a DetailView property that needs to be set in order to have the DetailView displayed on item level. This hierarchy can be compared to a PageControl. A page control has various pages (Item DetailView) that are shown when clicking on the Tab (Item). The container control that is responsible for displaying the DetailView is assigned to the TableView.

In the sample below when have dropped a TableView (TMSFMXNativeUITableView1) on the form along with a container view (TMSFMXNativeUIView1) and 3 addition controls that will be linked to the items (TMSFMXNativeUIDatePicker1, TMSFMXNativeUISlider1 and TMSFMXNativeUIButton1).

At designtime, this look similar as the image below.



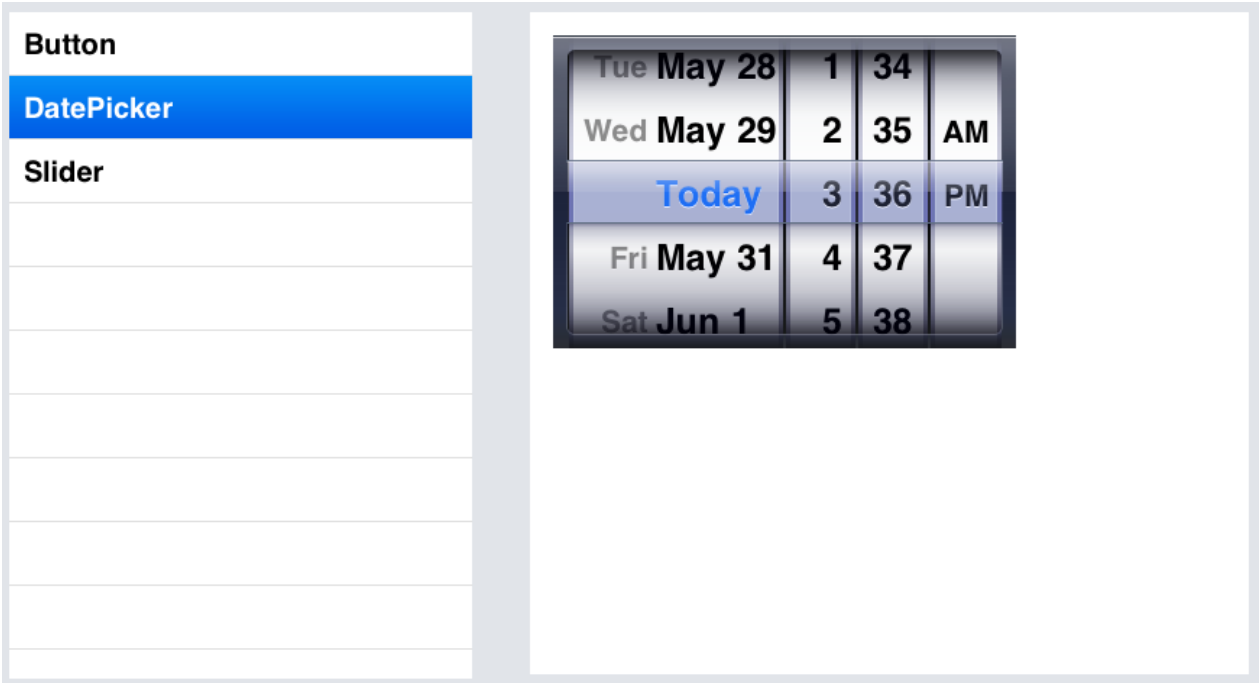
The code that links the items to the DetailView is shown below.

```
TMSFMXNativeUIDatePicker1.Visible := False;
TMSFMXNativeUIButton1.Visible := False;
TMSFMXNativeUISlider1.Visible := False;
TMSFMXNativeUITableView1.DetailView := TMSFMXNativeUIView1;
with TMSFMXNativeUITableView1.Sections.Add do
begin
  with Items.Add do
  begin
    Text := 'Button';
    DetailView := TMSFMXNativeUIButton1;
  end;
  with Items.Add do
  begin
    Text := 'DatePicker';
    DetailView := TMSFMXNativeUIDatePicker1;
  end;
  with Items.Add do
  begin
    Text := 'Slider';
    DetailView := TMSFMXNativeUISlider1;
  end;
```

```
end;
```

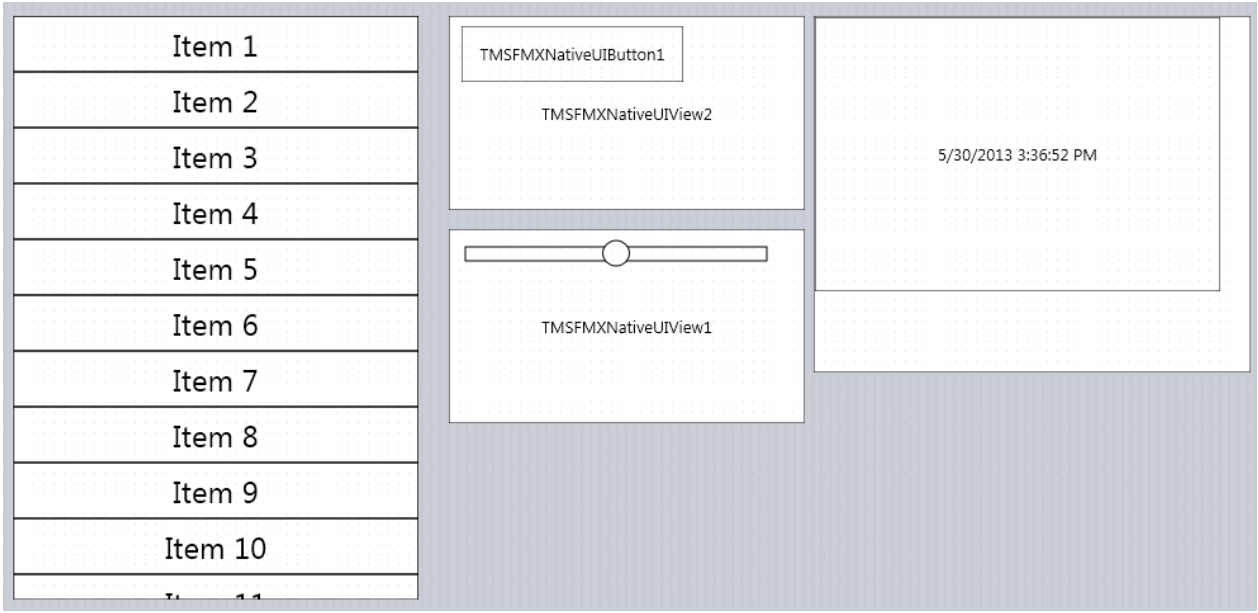
The container view is the TMSFMXNativeUIView1 and is linked to the DetailView property of the TMSFMXNativeUITableView1. The 3 children of the view each need to be assigned to the item's DetailView property and need to be set Visible false.

When running, the application will display an empty view and a TableView with 3 items. Clicking the items will display the correct DetailView in the container view.



The SubDetailView property has a similar purpose but this way of linking does not require an additional DetailView linked to the TableView. The SubDetailView is shown as a pushed detail from the main TableView. This is called Master-Detail. When changing the above sample so that the 3 children are set as SubDetailView, the container view can be removed. Each control is put in a TMSFMXNativeUIView instance as the view is stretched when it is pushed in the TableView.

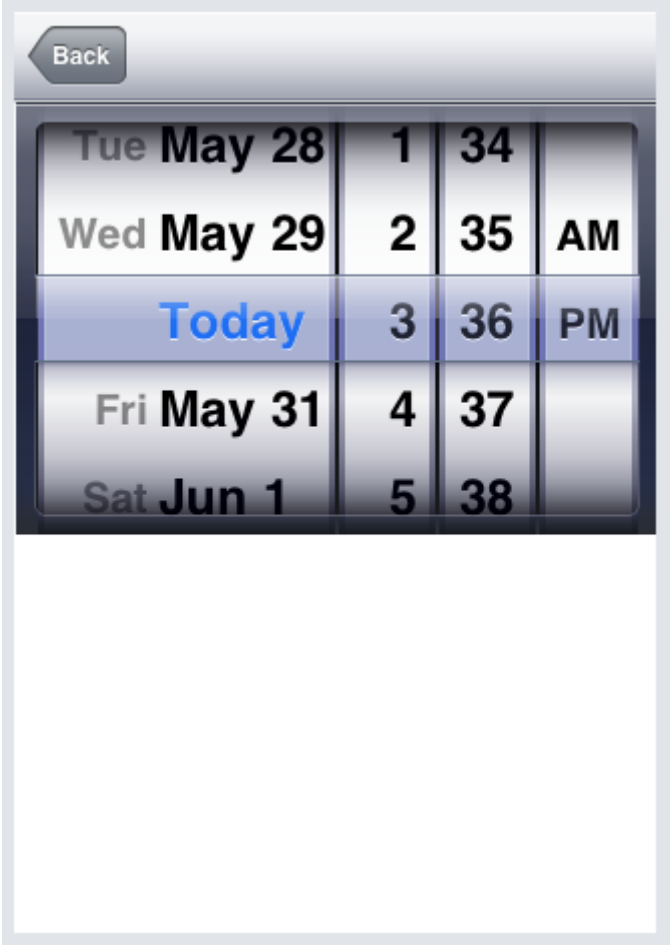
At designtime this looks similar like the image below.



The code for initialization:

```
TMSFMXNativeUIDatePicker1.Visible := False;
TMSFMXNativeUIView1.Visible := False;
TMSFMXNativeUIView2.Visible := False;
TMSFMXNativeUIView3.Visible := False;
TMSFMXNativeUITableView1.Options.ToolBar := True;
with TMSFMXNativeUITableView1.Sections.Add do
begin
  with Items.Add do
  begin
    Text := 'Button';
    SubDetailView := TMSFMXNativeUIView2;
  end;
  with Items.Add do
  begin
    Text := 'DatePicker';
    SubDetailView := TMSFMXNativeUIView3;
  end;
  with Items.Add do
  begin
    Text := 'Slider';
    SubDetailView := TMSFMXNativeUIView1;
  end;
end;
```

When clicking an item the correct DetailView is pushed in the TableView. To return to the main view, the ToolBar is enabled and automatically shows a back button.

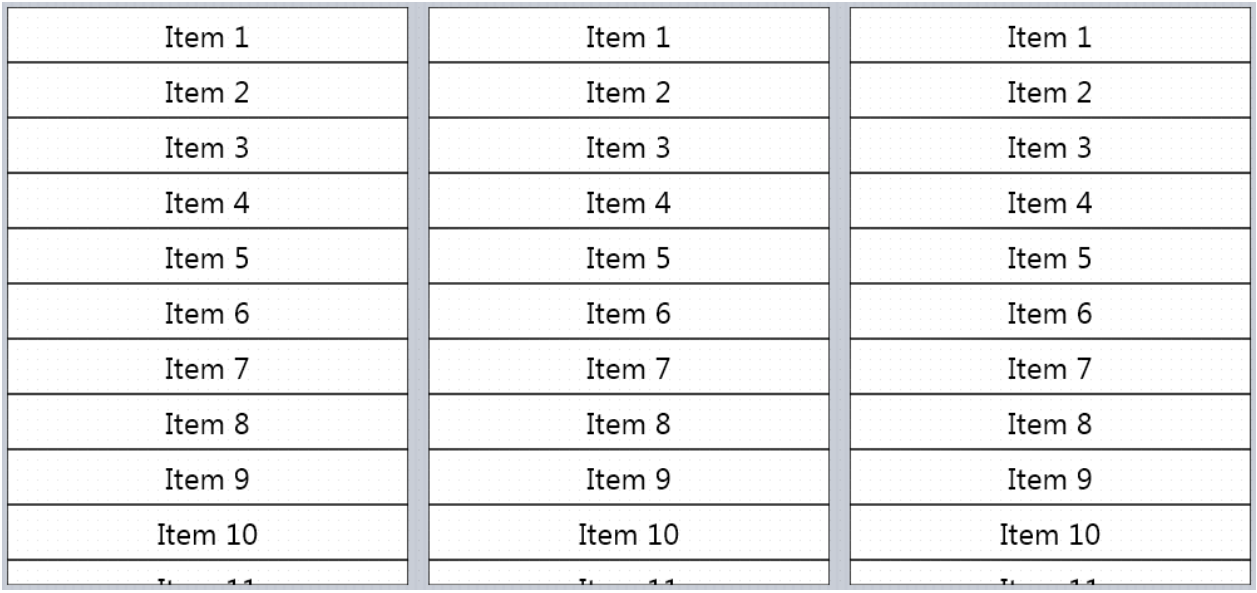


Master-Detail

Each TMS FMX Native UI Control can be used as a DetailView or SubDetailView of an item, so another instance of the TMSFMXNativeUITableView can be used and linked to the item. The second TableView also supports this type of linked thus the Master-Detail hierarchy can have multiple TMSFMXNativeUITableView instances linked to eachother and thus have multiple "levels" of detail.

This setup is similar as the one in the DetailView and SubDetailView chapter but requires an additional property to be set. This sample shows how to link 3 instances of TMSFMXNativeUITableView to eachother by means of a SubDetailView and shows the purpose of the MasterTableView property.

At designtime, we simply drop 3 instances of TMSFMXNativeUITableView on the form. The linking can be done at designtime, but is easier in code.



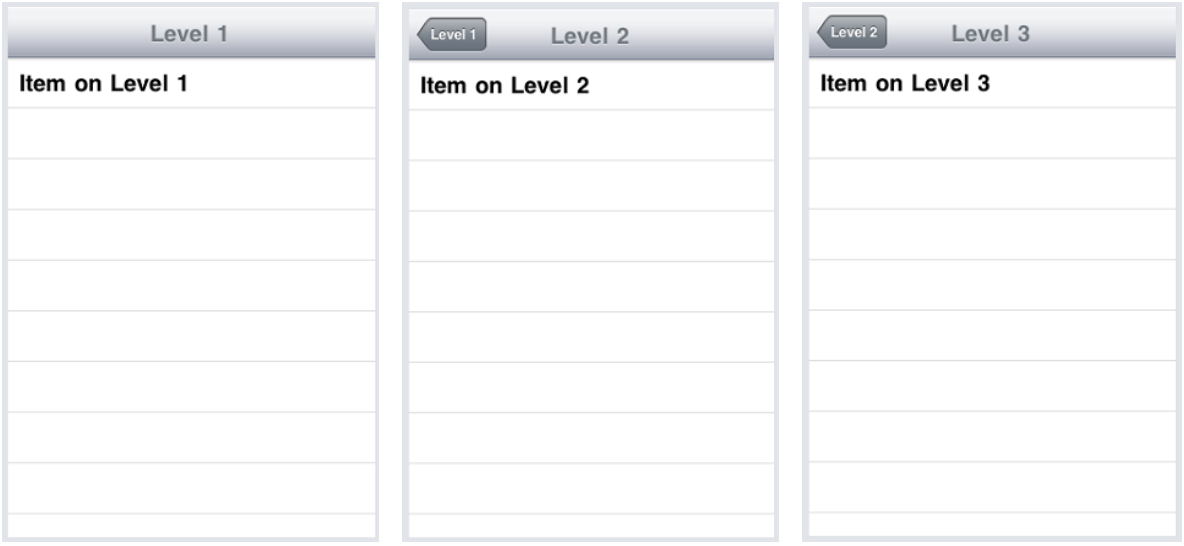
The code that accompanies this sample is shown below.

```
var
  s: TTMSFMXNativeUITableViewSection;
  it: TTMSFMXNativeUITableViewItem;
begin
  TMSFMXNativeUITableView1.Options.Header := 'Level 1';
  TMSFMXNativeUITableView2.Options.Header := 'Level 2';
  TMSFMXNativeUITableView3.Options.Header := 'Level 3';

  TMSFMXNativeUITableView1.Options.ToolBar := True;
  TMSFMXNativeUITableView2.MasterTableView :=
TMSFMXNativeUITableView1;
  TMSFMXNativeUITableView3.MasterTableView :=
TMSFMXNativeUITableView1;
  TMSFMXNativeUITableView2.Visible := False;
  TMSFMXNativeUITableView3.Visible := False;

  s := TMSFMXNativeUITableView1.Sections.Add;
  it := s.Items.Add;
  it.Text := 'Item on Level 1';
  it.SubDetailView := TMSFMXNativeUITableView2;

  s := TMSFMXNativeUITableView2.Sections.Add;
  it := s.Items.Add;
  it.Text := 'Item on Level 2';
  it.SubDetailView := TMSFMXNativeUITableView3;
```

```
s := TMSFMXNativeUITableView3.Sections.Add;
it := s.Items.Add;
it.Text := 'Item on Level 3';
```

When starting the application, only the first TableView is visible, when clicking on the item, the second TableView is shown and clicking on the item of the second TableView shows the third TableView. Notice that we have only enabled the ToolBar on the first TableView as this takes care of displaying the position in the hierarchy. Therefore the MasterTableView property is necessary as the first TableView needs to know which sub-TableView instances are linked.

When navigating, the back button is automatically updated and the header of the TableView is set. Clicking on the back button returns one step in the hierarchy.



As the items are fixed, the items on each TableView will remain the same even if there are multiple items on the first level that all link to the same sub-TableView. When clicking on an item, the OnBeforeShowDetailView is called. This event can be used to customize the items that are displayed per level, based on the previous level. This event is available per TableView.

Virtual Mode

The previous samples are all based on a collection that needs to be filled with sections and items. The TableView also supports a virtual mode where items can be displayed without a collection. This can be of use when fetching data from a Database, a custom collection or list that is maintained in the application and there is no need to map data to the properties of the built-in section and items collection. The 4 events for minimal implementation in virtual mode are the OnGetNumberOfSections, the OnGetNumberOfRowsInSection, the OnGetTitleForHeaderInSection and the OnGetItemText. Below is a sample that implements these events and shows 2 sections with a couple of items.

```
procedure TForm1.TMSFMXNativeUITableView1GetItemText(Sender:
TObject;
  ASection, ARow: Integer; var AText: string);
begin
  AText := 'S ' + inttostr(ASection) + ' Item ' + inttostr(ARow);
end;

procedure TForm1.TMSFMXNativeUITableView1GetNumberOfRowsInSection(
  Sender: TObject; ASection: Integer; var ANumberOfRows: Integer);
begin
  case ASection of
    0:  ANumberOfRows := 3;
    1:  ANumberOfRows := 5;
  end;
end;

procedure TForm1.TMSFMXNativeUITableView1GetNumberOfSections(Sender:
TObject;
  var ANumberOfSections: Integer);
begin
  ANumberOfSections := 2;
end;

procedure TForm1.TMSFMXNativeUITableView1GetTitleForHeaderInSection(
  Sender: TObject; ASection: Integer; var ATitle: string);
begin
  ATitle := 'Section ' + inttostr(ASection);
end;
```

Custom Collection

If the data structure of your application contains extra properties that needs to be displayed in the item and the properties of the base collection are not sufficient, the TableView exposes 2 virtual functions that can be overridden to add additional properties. The CustomCells demo that is included in the distribution makes use of a TMSFMXNativeUITableViewMail instance that inherits from the TMSFMXNativeUITableView and overrides the virtual functions that create the section and item collection. When examining the code of this class, you will notice that the TMSFMXNativeUITableViewMailItem collection item class is used to add additional properties such as Sender, Date, Title, Description and Unread.

The TMSFMXNativeUITableViewMail inherits from TMSFMXNativeUITableView and overrides the CreateSections function that returns the custom section collection TMSFMXNativeUITableViewMailSections. As each section has an items collection, the CreateItems function needs to be overridden to return the TMSFMXNativeUITableViewMailItems collection that holds the customized items.

The creation of a custom collection can be sufficient to persist non-visual data in your application. If you need additional visual representation in an item, then the Custom Items chapter will explain how you can override the default layout of a TableView item and display custom data.

The Source of the TMSFMXNativeUITableView can be found below in the “Resources” chapter

Custom Items

The TableView has a few predefined styles for an item that position the image, title and description on fixed positions. Using the default or changing the style of an item can be sufficient for your application. If the style of an item is not sufficient for your application, the TableView exposes 2 additional public procedures and events that can be implemented to customize your TableView item layout.

As mentioned in the chapter “Custom Collection”, the TMSFMXNativeUITableViewMail implementation overrides the default collection and provides properties to persist additional data in your application.

The TMSFMXNativeUITableViewMail implementation also demonstrates how to use the 2 virtual procedures that are called when an item is being created and displayed. The DoCreateCell is called when a new item (cell) is being created. This procedure, and the event OnCreateCell, can be used to add additional controls to your item layout that can linked to the properties in the custom collection. These events can also be used to customize your application with the default collection as well, but in this sample the combination of a custom collection and a custom item layout is a typical scenario in real application.

When investigating the source of the TMSFMXNativeUITableViewMail (found in the “Resources” chapter below) you will notice that custom label instances are added and customized linking to the custom collection.

TMSFMXNativeUIToolBar

Usage

A TMSFMXNativeUIToolBar is a control that displays one or more Buttons, called toolbar items.

Published Properties

Items	The items displayed on the ToolBar.
Items[Index] → Action	Property to assign an action combined with an action list.
Items[Index] → Bitmap	The bitmap used inside an item.
Items[Index] → CustomView	Custom view of an item used to display content from another TMS FMX Native UI Control linked to an item within the toolbar, when the Kind property is set to ikCustom.
Items[Index] → Enabled	Enables / disables an item.
Items[Index] → Kind	The kind of item that is display in the ToolBar, an item can be a normal, system or custom item.
Items[Index] → Style	The style of an item, applied when using normal or system Kind. The style can be plain, done or bordered.
Items[Index] → SystemItem	When setting the kind to ikSystem, the SystemItem property determines which icon is display inside the button.
Items[Index] → Text	The text of a button.
Items[Index] → Visible	Shows / hides a button.
Style	The ToolBar style that specifies its appearance.
TintColor	The color used to tint the bar.
Translucent	A Boolean value that indicates whether the ToolBar is translucent or not.

Visible	Shows / hides the ToolBar.

Public Properties

ToolBar	Returns a reference to the native iOS UIToolbar.

Methods

BeginUpdate / EndUpdate	Wrapping code to block direct updates to the ToolBar. This is done for performance when loading a large amount of items (buttons).
FindItemByControl(AItem: UIBarButtonItem): TTMSFMXNativeUIToolBarItem;	Returns the item by passing a reference to the native UIBarButtonItem that is linked to an item.

Events

OnItemClick	Event called when clicking on an item (button)

TMSFMXNativeUIPickerView

Usage

The TMSFMXNativeUIPickerView class implements columns, that use a spinning-wheel or slot-machine metaphor to show one or more sets of values. Users select values by rotating the wheels so that the desired row of values aligns with a selection indicator.

Published Properties

Columns	Collection of columns used in the PickerView.
Columns → Items	Collection of items per column.
Columns → Items → Text	The text of an item per column.
ShowSelectionIndicator	Shows the selection indicator that overlaps the columns. The selection of an item is always displayed in the center of the control.
Visible	Shows / hides the PickerView.

Public Properties

PickerView	Returns a reference to the native iOS UIPickerView.

Methods

SelectRowInColumn(ARow, AColumn: Integer; AAnimated: Boolean);	Selects a specific row (item) in a specific column.

SelectedRowForColumn(AColumn: Integer): Integer;	Returns the selected index of a row (item) in a specific column.

Events

OnGetNumberOfColumns	Returns the number of columns.
OnGetNumberOfRowsForColumn	Returns the number of rows (items) for a column.
OnGetTitleForRow	Returns the title for a specific row at a specific column.
OnValueChanged	Event called when a value of a specific column has changed.

TMSFMXNativeUIDatePicker

Usage

The TMSFMXNativeUIDatePicker class implements an object that uses multiple rotating wheels to allow users to select dates and times. iPhone examples of a date picker are the Timer and Alarm (Set Alarm) panes of the Clock application. You may also use a date picker as a countdown timer.

Published Properties

CountDownDuration	This property (in seconds) is only used to display a number when the Mode is set to dpmDatePickerModeCountDownTimer. It does not actually count down. This requires manual implementation.
DateTime	The datetime displayed by the DatePicker.
MaximumDateTime	The maximum datetime that a DatePicker can show.
MinimumDateTime	The minimum datetime that a DatePicker can show.
MinuteInterval	The interval at which the DatePicker should display minutes.
Mode	The value of this property indicates the mode of a DatePicker. It determines whether the DatePicker allows selection of a date, a time, both date and time, or a countdown time. The default mode is dpmDatePickerModeDateAndTime.

Visible	Shows / hides the DatePicker.

Public Properties

DatePicker	Returns a reference to the native iOS UIDatePicker.

Methods

NSDateToDate(ADateTime: NSDate): TDateTime;	Returns a TDateTime from a native iOS NSDate instance.

Events

OnValueChanged	Event called when a date / value has changed.

Countdown timer

With the Mode property set to dpmDatePickerModeCountDownTimer the DatePicker can be set to a countdown timer. Additionally the amount of seconds to countdown from needs to be set with the CountDownDuration property.

In the sample below, the DatePicker is set to a countdown duration of 60 seconds. The user can select an amount of countdown seconds from which to start from.

```
TMSFMXNativeUIDatePicker1.Mode := dpmDatePickerModeCountDownTimerl;
TMSFMXNativeUIDatePicker1.CountDownDuration := 60;
FDuration := 60;
```

The OnValueChanged event is triggered when the user changes the value on the countdown wheel.

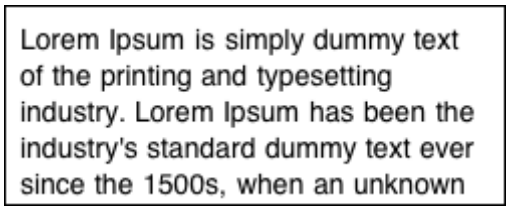
```
procedure TForm1.TMSFMXNativeUIDatePicker1ValueChanged(ASender: TObject;
  ADateTime: TDateTime);
begin
  FDuration := TMSFMXNativeUIDatePicker1.CountDownDuration;
end;
```

To actually use it as a timer, a TTimer component is used to start counting down from the value chosen by the user. The timer can be used to, as an example, update a label.

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
```

```
  FDuration := FDuration - 1;
  TMSFMXNativeUILabel1.Text := 'Seconds left = ' + floattostr(FDuration);
end;
```

TMSFMXNativeUITextView



Usage

The TMSFMXNativeUITextView class implements the behavior for a scrollable, multiline text region. The class supports the display of text using custom style information and also supports text editing. You typically use a text view to display multiple lines of text, such as when displaying the body of a large text document.

Published Properties

Alignment	The technique to use for aligning the text.
Editable	A Boolean value indicating whether the TextView is editable or not.
Font	Specifies the Font name and Size of the TextView.
Text	The Text of the TextView.
TextColor	The color of the TextView.
TextInputTraits	The TextInputTraits property defines features that are associated with keyboard input.
TextInputTraits → AutoCapitalizationType	This property determines at what times the Shift key is automatically pressed, thereby making the typed character a capital letter. The default value for this property is actTextAutocapitalizationTypeSentences.
TextInputTraits → AutoCorrectionType	This property determines whether auto-correction is enabled or disabled during typing.
TextInputTraits → SpellCheckingType	This property determines whether spell-checking is enabled or disabled during typing. With spell-checking enabled, the text object generates red underlines for all misspelled words.
TextInputTraits →	A Boolean value indicating whether the return key is automatically enabled when text is

EnablesReturnKeyAutomatically	entered by the user.
TextInputTraits → KeyBoardAppearance	The appearance style of the keyboard that is associated with the TextView.
TextInputTraits → KeyBoardType	The keyboard style associated with the TextView.
TextInputTraits → ReturnKeyType	The contents of the "return" key.
TextInputTraits → SecureTextEntry	Identifies whether the TextView should hide the text being entered.
Visible	Shows / hides the TextView.

Public Properties

TextView	Returns a reference to the native iOS UITextView.

Events

OnChanged	Event called when the value of the TextView has changed.
OnDidBeginEditing	Event called when editing did begin.
OnDidChangeSelection	Event called when selection of the text did change.
OnDidEndEditing	Event called when editing did end.
OnShouldBeginEditing	Event called when editing should begin.
OnShouldChangeTextInRange	Event called when a specified text in range should be changed. The TextView calls this event whenever the user types a new character or deletes an existing character. Implementation of this method is optional. You can use this method to replace text before it is committed to the TextView storage. For example, a spell checker might use this method to replace a misspelled word with the correct spelling.
OnShouldEndEditing	Event called when editing should end.

TMSFMXNativeUILabel

Usage

The TMSFMXNativeUILabel class implements a read-only text view. You can use this class to draw one or multiple lines of static text, such as those you might use to identify other parts of your user interface.

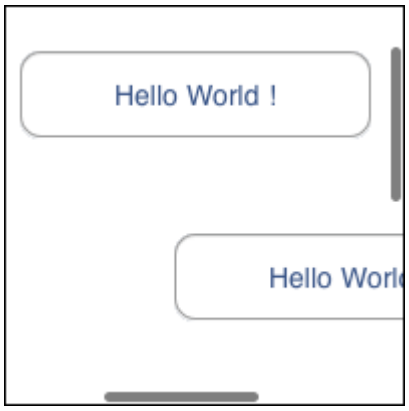
Published Properties

Alignment	The text alignment of the label.
Color	The background color of the label.
Font	Specifies the font name and size of the Label.
LineBreakMode	The technique to use for wrapping and truncating the Label's text.
NumberOfLines	The number of lines that are allowed to be displayed inside the Label. The default value is 1. For Linbreak to work, NumberOfLines need to be larger than 1.
TextColor	The color of the text.
Visible	Shows / hides the Label.

Public Properties

Lbl	Returns a reference to the native iOS UILabel.

TMSFMXNativeUIScrollView



Usage

The TMSFMXNativeUIScrollView class provides support for displaying content that is larger than the size of the application's window. It enables users to scroll within that content by making swiping gestures.

Published Properties

AlwaysBounceHorizontal	A Boolean value that determines whether bouncing always occurs when horizontal scrolling reaches the end of the content view or not.
AlwaysBounceVertical	A Boolean value that determines whether bouncing always occurs when vertical scrolling reaches the end of the content or not.
Bounces	Bounces the view horizontally or vertically depending on the AlwaysBounceHorizontal and AlwaysBounceVertical properties.
DirectionalLockEnabled	A Boolean value that determines whether scrolling is disabled in a particular direction or not.
Enabled	A Boolean value that determines whether scrolling is enabled or not.
ShowsHorizontalScrollIndicator	A Boolean value that controls whether the horizontal scroll indicator is visible or not.
ShowsVerticalScrollIndicator	A Boolean value that controls whether the vertical scroll indicator is visible or not.

ViewForZooming	View of the ScrollView used for zooming.
Visible	Shows / hides the ScrollView.

Public Properties

ScrollView	Returns a reference to the native iOS UIScrollView.

Events

OnViewForZoomingInScrollView	Returns a reference to a TMS FMX Native iOS Control used for zooming.

TMSFMXNativeUIProgressView

Usage

You use the TMSFMXNativeUIProgressView class to depict the progress of a task over time.

Published Properties

Progress	The current progress of the ProgressView. The progress is a single value between 0.0 and 1.0.
Style	The graphical style of the ProgressView.
Visible	Shows / hides the ProgressView.

Public Properties

ProgressView	Returns a reference to the native iOS UIProgressView.

Events

OnValueChanged	Event called when the value of the ProgressView has changed.

TMSFMXNativeUISegmentedControl

Usage

A UISegmentedControl object is a horizontal control made of multiple segments, each segment functioning as a discrete Button. A segmented control affords a compact means to group together a number of controls.

Published Properties

Items (Segments)	Collection of items (segments) in the SegmentedControl.
Items → Bitmap	The bitmap of an item (segment).
Items → Enabled	Sets an item (segment) enabled or not.
Items → Text	The text of an item (segment) in case there is no bitmap assigned.
Style	The style of the SegmentedControl.
SelectedSegmentIndex	Sets or gets the SelectedSegmentIndex which is the index of an item in the Items collection.
TintColor	The tint color of the SegmentedControl.
Visible	Shows / hides the SegmentedControl.

Public Properties

Button	Returns a reference to the native iOS UISegmentedControl.

Methods

BeginUpdate / EndUpdate	Wrapping code to block direct updates to the SegmentedControl. This is done for performance when loading a large amount of items and

	content.

Events

OnValueChanged	Event called when the selected segment index has changed.

TMSFMXNativeUIStepper

Usage

A TMSFMXNativeUIStepper control provides a user interface for incrementing or decrementing a value. A stepper displays two Buttons, one with a minus ("-") symbol and one with a plus ("+") symbol.

Published Properties

AutoRepeat	If true, the user pressing and holding on the stepper repeatedly alters value.
Continuous	If true, value change events are sent immediately when the value changes during user interaction. If false, a value change event is sent when user interaction ends.
MaximumValue	The highest possible numeric value for the Stepper.
MinimumValue	The lowest possible numeric value for the Stepper.
StepValue	The step, or increment, value for the Stepper.
Value	The value of the Stepper.
Visible	Shows / hides the Stepper.
Wraps	If true, incrementing beyond maximumValue sets value to minimumValue; likewise, decrementing below minimumValue sets value to maximumValue. If false, the Stepper does not increment beyond maximumValue nor does it decrement below minimumValue but rather holds at those values.

Public Properties

Stepper	Returns a reference to the native iOS UIStepper.

Events

OnValueChanged	Event called when the value of the stepper has changed.

TMSFMXNativeUITextField

Usage

A TMSFMXNativeUITextField object is a control that displays editable text and sends an action message to a target object when the user presses the return Button. You typically use this class to gather small amounts of text from the user and perform some immediate action, such as a search operation, based on that text.

Published Properties

Alignment	The alignment of the text.
BorderStyle	The border style of the TextField.
ClearButton	Shows / hides a clear button on the TextField.
TextColor	The text color of the text.
TextInputTraits	The TextInputTraits property defines features that are associated with keyboard input.
TextInputTraits → AutoCapitalizationType	This property determines at what times the Shift key is automatically pressed, thereby making the typed character a capital letter. The default value for this property is actTextAutocapitalizationTypeSentences.
TextInputTraits → AutoCorrectionType	This property determines whether auto-correction is enabled or disabled during typing.
TextInputTraits → SpellCheckingType	This property determines whether spell-checking is enabled or disabled during typing. With spell-checking enabled, the text object generates red underlines for all misspelled words.
TextInputTraits → EnablesReturnKeyAutomatically	A Boolean value indicating whether the return key is automatically enabled when text is entered by the user.
TextInputTraits → KeyBoardAppearance	The appearance style of the keyboard that is associated with the TextField.
TextInputTraits → KeyBoardType	The keyboard style associated with the TextField.

TextInputTraits → ReturnKeyType	The contents of the "return" key.
TextInputTraits → SecureTextEntry	Identifies whether the TextField should hide the text being entered.
Visible	Shows / hides the TextField.

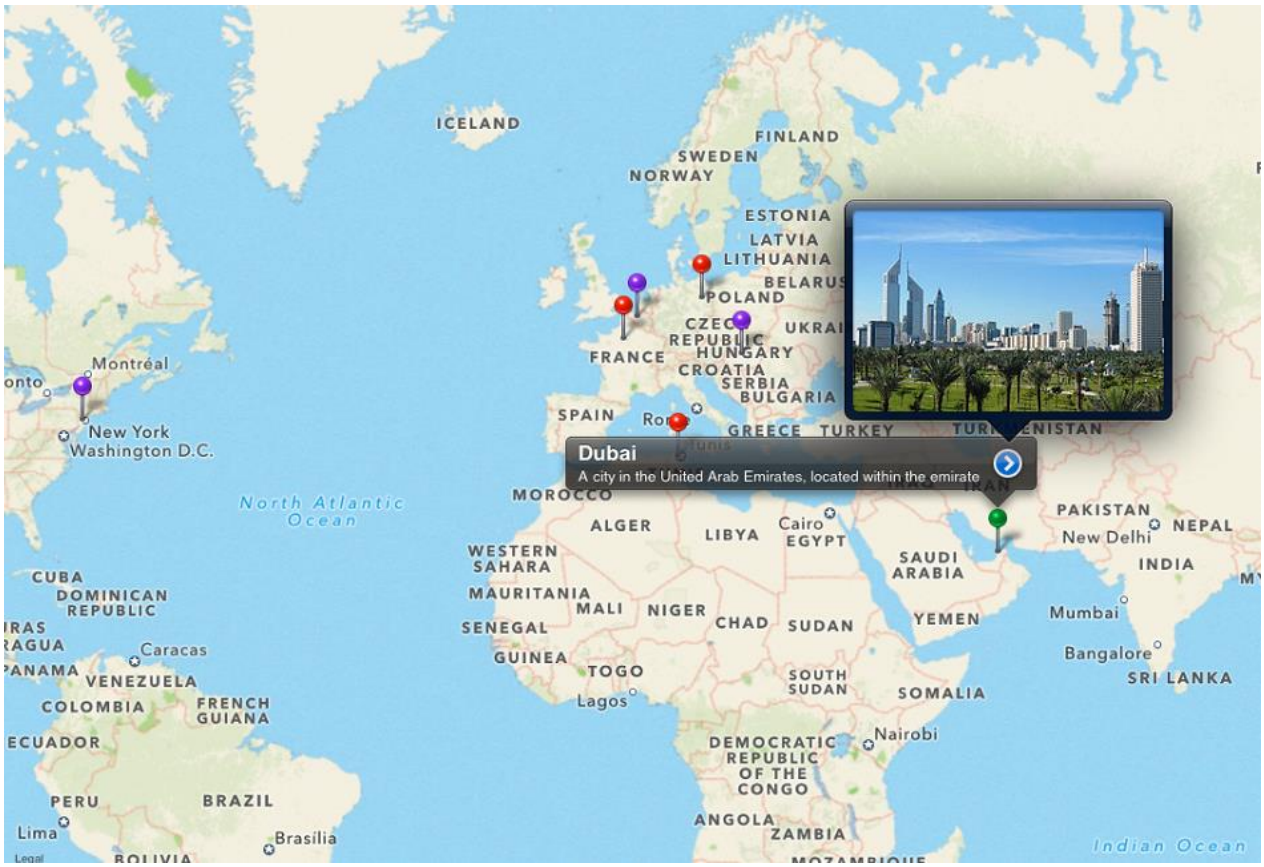
Public Properties

TextField	Returns a reference to the native iOS UITextField.

Events

OnChanged	Event called when the text of the TextField has changed.
OnDidBeginEditing	Event called when editing did begin.
OnDidChangeSelection	Event called when selection of the text did change.
OnDidEndEditing	Event called when editing did end.
OnShouldBeginEditing	Event called when editing should begin.
OnShouldChangeTextInRange	Event called when a specified text in range should be changed. The TextView calls this event whenever the user types a new character or deletes an existing character. Implementation of this method is optional. You can use this method to replace text before it is committed to the TextView storage. For example, a spell checker might use this method to replace a misspelled word with the correct spelling.
OnShouldClear	Event called if the current text should be cleared.
OnShouldEndEditing	Event called when editing should end.

TMSFMXNativeMKMapView



Usage

A TMSFMXNativeMKMapView object provides an embeddable map interface, similar to the one provided by the 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.

Published Properties

Annotations	A collection of annotations, used to annotate the map and display custom information through annotations pins.
Annotations[Index] → AnimatesDrop	Animates a drop of a pin when added to the MapView.
Annotations[Index] → Bitmap	Shows an image for the view of an annotation. If a bitmap is assigned, the default pin is replaced with this image.

Annotations[Index] → CanShowCallout	Enables or disables showing the default callout for an annotation, which shows a title, subtitle and / or callout accessory views.
Annotations[Index] → Draggable	Enables or disables dragging of an annotation. When dragging an annotation the OnMapViewAnnotationDragStateChanged event is called with various drag states.
Annotations[Index] → Enabled	Enables or disables interaction with an annotation.
Annotations[Index] → LeftCalloutAccessoryView	The left callout accessory view of an annotation. This view can be linked to another instance of a TMS FMX Native UI Control.
Annotations[Index] → PinColor	The color of the pin of an annotation.
Annotations[Index] → RightCalloutAccessoryView	The right callout accessory view of an annotation. This view can be linked to another instance of a TMS FMX Native UI Control.
Annotations[Index] → SubTitle	The sub title of an annotation shown in the callout.
Annotations[Index] → Title	The title of an annotation shown in the callout. If the title is empty, no callout is shown.
MayType	The type of data displayed by the MapView.
ScrollEnabled	Enables / disables scrolling on the MapView.
ShowsUserLocation	Shows / hides the current user location.
Visible	Shows / hides the MapView.
ZoomEnabled	Enables / disables zooming on the MapView.

Public Properties

MapView	Returns a reference to the native iOS MKMapView.

Methods

AddAnnotation(ALatitude, ALongitude: Double; ATitle, ASubTitle: String):	Adds and returns a new annotation based on a Latitude, Longitude, Title and SubTitle.

TTMSFMXNativeMKAnnotation	
AddAnnotation(ALocation: TTMSFMXNativeMKMapLocation; ATitle, ASubTitle: String): TTMSFMXNativeMKAnnotation	Adds and returns a new annotation based on a location, Title and SubTitle.
AddAnnotation(ALocation: TTMSFMXNativeMKMapLocation): TTMSFMXNativeMKAnnotation	Adds and returns a new annotaton based on a location.
AddAnnotation(ALatitude, ALongitude: Double): TTMSFMXNativeMKAnnotation	Adds and returns a new annotation based on a Latitude and Longitude.
GetAnnotation(Annotation: MKAnnotion): TTMSFMXNativeMKAnnotation	Returns the annotation collection item based on the native MKAnnotation.
DeSelectAnnotation(AAnnotation: TTMSFMXNativeMKAnnotation)	Deselects the annotation with or without animation.
GetAnnotation(AnnotationView: MKAnnotationView): TTMSFMXNativeMKAnnotation	Returns the annotation collection item based on the native MKAnnotationView.
GetUserLocation: TTMSFMXNativeMKMapLocation	Returns the current user location.
GetRegion: TTMSFMXNativeMKMapRegion;	Returns the current region.
RemoveAllAnnotations	Removes all annotations from the collection and the MapView.
RemoveAnnotation(AAnnotation: TTMSFMXNativeMKAnnotation)	Removes a specific annotation from the collection and the MapView.
SelectAnnotaion(AAnnotation: TTMSFMXNativeMKAnnotation; AAnimated: Boolean)	Selects a specific annotation and shows the callout, with or without animation.
SetCenterLocation(ALocation: TTMSFMXNativeMKMapLocation)	Centers the map at a specific location.
SetRegion(ARegion: TTMSFMXNativeMKMapRegion; AAnimated: Boolean)	Sets the visible region of the MapView.
SetRegion(ATopLeftLocation, ABottomRightLocation: TTMSFMXNativeMKMapLocation; AAnimated:	Sets the visible region of the MapView with TopLeft and BottomRight coordinates.

Boolean)	
XYToCoordinate(X, Y: Single): TTMSFMXNativeMKMapLocation	Returns a latitude and longitude of an X and Y coordinate on the map based on the current region of the MapView.
ZoomToFitAnnotations(AIncludeUserLocation: Boolean = false; ALatitudeSpanOffset: Double = 0; ALongitudeSpanOffset: Double = 0);	Zooms the mapview to show/fit all annotations with optional parameters to include user location and additional region span offset.

Events

OnAnnotationDragStateChanged	Event called when a pin is being dragged to a new location.
OnAnnotationLeftCalloutAccessoryTapped	Event called when the left callout accessory is tapped on an annotation.
OnAnnotationRightCalloutAccessoryTapped	Event called when the right callout accessory is tapped on an annotation.
OnClick	Event called when clicking on the map. The latitude and longitude of the position on the map are passed as parameters.
OnDidDeselectAnnotationView	Event called when deselectin an annotation.
OnDidFailLoadingMap	Event called when the map loading failed.
OnDidFailToLocateUser	Event called when the map did fail to locate the user location when the user location is active.
OnDidFinishLoadingMap	Event called when the map did finish loading.
OnSelectAnnotationView	Event called when an annotation is selected.
OnDidStopLocatingUser	Event called when the MapView stops locating the user.
OnDidUpdateUserLocation	Event called when the user location is updated.
OnLongPress	Event called when tap holding on the MapView for at least 1.5 seconds.
OnRegionDidChangeAnimated	Event called when the region of the MapView is changed.

OnRegionWillChangeAnimated	Event called when the region of the MapView will change.
OnWillStartLoadingMap	Event called when the MapView will start loading.
OnWillStartLocatingUser	Event called when the MapView will start locating the user.

Adding Annotations

Annotations can be added to the map by directly adding them to the collection, or through one of the AddAnnotation overload methods. Below is a sample that demonstrates this.

In this sample, we drop a TMSFMXNativeMKMapView control on the form and add the following code in a button click:

```
var
  loc: TTMSFMXNativeMKMapLocation;
begin
  loc :=
TMSFMXNativeMKMapView1.XYToCoordinate(TMSFMXNativeMKMapView1.Width /
2, TMSFMXNativeMKMapView1.Height / 2);
  TMSFMXNativeMKMapView1.AddAnnotation(loc, 'Hello World',
'Subtitle');
```

This code will return the correct coordinate of the center of the map, regardless of where the map is positioned. Clicking the button drops an annotation on the MapView:

If you pan the Map, and click the button, the pin will be dropped in the center of the MapView again, but on a different coordinate. The XYToCoordinate functionality is using the current region, and zooming level of the MapView to return the correct coordinate.

Pin vs View

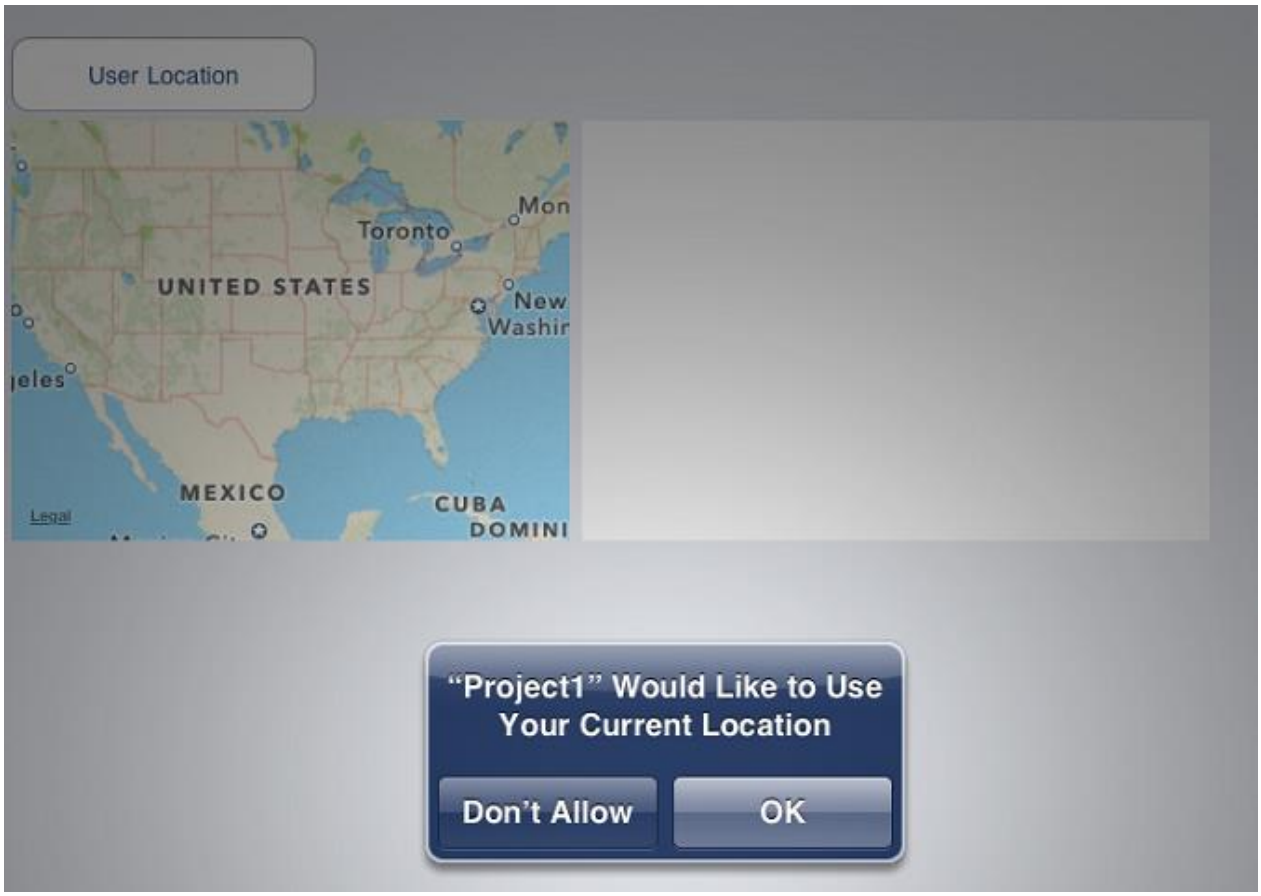
By default, the annotation that is added on the map displays a pin, but this can also be changed per annotation. On the annotation collection item, a Bitmap property is available to customize the default view of an annotation. The sample below shows how to add a custom image for an annotation.

```
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');
```

User Location

The MapView can also display the user location, to show the user location, you can set the property ShowsUserLocation to true. When the application shows the user location for the first time, the application asks if it is ok to allow access. After clicking ok, the user location is being displayed in the MapView.

When the user location is displayed, the MapView does not automatically scroll to the location. The sample implements the OnDidUpdateUserLocation to center the user location and log the latitude and longitude in a listbox.



Included in the distribution is a Map demo that demonstrates adding annotations, panning and zooming in the MapView as well as showing a callout accessory view to display additional information.

TMSFMXNativeFMXWrapper

Usage

The TMSFMXNativeFMXWrapper is a wrapper component that is able to display a separate form as a subview of another control. The wrapper component can, for example, be used in the TMSFMXNativeUITableView as a detailview or subdetailview.

Published Properties

Visible	Shows / hides the wrapper FMX form.
Form	Property to assign a Form to the wrapper. The wrapper then displays the content of the form as a subview of the wrapper view.

TMSFMXNativeUIImageView

Usage

A TMSFMXNativeUIImageView object provides a view-based container for displaying a single image. The TMSFMXNativeUIImageView supports following image formats:

Tagged Image File Format (TIFF)

.tiff, .tif

Joint Photographic Experts Group (JPEG)

.jpg, .jpeg

Graphic Interchange Format (GIF)

.gif

Portable Network Graphic (PNG)

.png

Windows Bitmap Format (DIB)

.bmp, .BMPf

Windows Icon Format

.ico

Windows Cursor

.cur

XWindow bitmap

.xbm

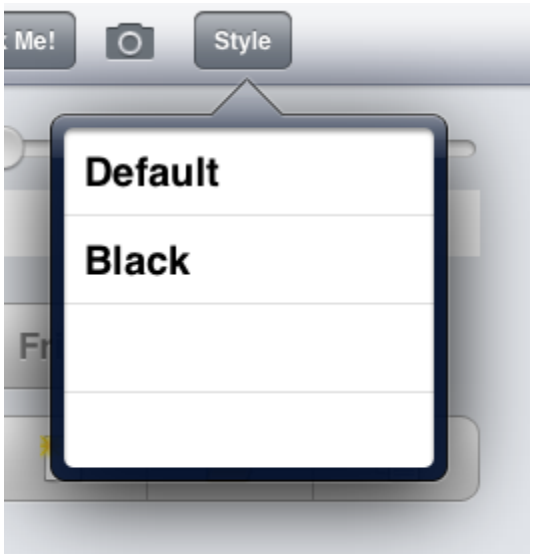
Properties

Bitmap	Sets the image of an ImageView.
BitmapFile	A direct link to an image file located in the root or documents directory.
BitmapLink	A link to another TBitmap instance which can be used multiple times to save resources.
ContentMode	The way the image is displayed inside the boundaries of the control, with various options such as aspect ratio, stretching and centering.
Visible	Shows / hides the ImageView.

Public Properties

ImageView	Returns a reference to the native iOS UIImageView.

TMSFMXNativeUIPopoverController



Usage

The TMSFMXNativeUIPopoverController class is used to manage the presentation of content in a popover. You use popovers to present information temporarily but in a way that does not take over the entire screen like a modal view does. The popover content is layered on top of your existing content in a special type of window. The popover remains visible until the user taps outside of the popover window or you explicitly dismiss it. Popover controllers are for use exclusively on iPad devices. Attempting to create an instance of the TMSFMXNativeUIPopoverController on devices other than an iPad results in an exception.

Published Properties

View	View of the PopOver used to display content from another TMS FMX Native UI Control in a popup.

Public Properties

PopOver	Returns a reference to the native iOS UIPopoverController.

Methods

ShowFromButton(AButton: UIBarButtonItem)	Shows the popup from a ToolBar button. (iPad only)

ShowFromControl(AControl: TTMSFMXNativeUIBaseControl)	Shows the popup from a TMS FMX Native UI Control placed on the form. (iPad only)
ShowFromRect(ARect: TRectF)	Shows the popup from a specific rectangle in the main view. (iPad only)
ShowFromRectInView(ARect: TRectF; AView: UIView)	Shows the popup from a specific rectangle in a specific view. (iPad only)

TMSFMXNativeUIView

Usage

The UIView class defines a rectangular area on the screen and can contain multiple other controls.

Published Properties

Visible	Shows / hides the View.

Public Properties

View	Returns a reference to the native iOS UIView.

Published Events

OnDrawRect	Event to perform custom drawing inside the View.

TMSFMXNativeUIImagePickerController

Usage

The TMSFMXNativeUIImagePickerController manages customizable, system-supplied user interfaces for taking pictures and movies on supported devices, and for choosing saved images and movies for use in your application.

Published Properties

AllowsEditing	Allows editing of images after selecting an image.
CameraCaptureMode	Sets the camera to photo or video capture mode.
CameraDevice	Determines whether the front or rear camera should be used.
CameraFlashMode	The flash mode of the camera set to off, on or auto.
EditedImage	Boolean to determine if the ImagePicker needs to save the original or edited image. Editing an image can be done when AllowsEditing is true.
SaveToAlbum	When an image is taken, automatically save it to the users album.
ShowCameraControls	Shows / hides the camera controls when the source type is set to stImagePickerControllerSourceTypeCamera.
SourceType	Specific whether the image picker controller should display the photo library, the camera or the saved photos album.
VideoMaximumDuration	Sets the maximum duration of a video.
VideoQuality	Sets the quality of a video.

Public Properties

ImagePicker	Returns a reference to the native iOS UIImagePickerController.

Popover	Returns a reference to the native iOS UIPopoverController.

Methods

Hide	Hides the ImagePicker.
Show	Shows the ImagePicker fullscreen. (iPhone & iPad), on iPad only when SourceType property is set to stImagePickerControllerSourceTypeCamera. When SourceType is set to a different value, use the other Show methods such as ShowFromButton or ShowFromControl.
ShowFromButton(AButton: UIBarButtonItem)	Shows the ImagePicker from a ToolBar button. (iPad only)
ShowFromControl(AControl: TTMSFMXNativeUIBaseControl)	Shows the ImagePicker from a TMS FMX Native UI Control on the form. (iPad only)
ShowFromRect(ARect: TRectF)	Shows the ImagePicker from a specific rectangle in the main view. (iPad only)
ShowFromRectInView(ARect: TRectF; AView: UIView)	Shows the ImagePicker from a specific rectangle in a specific view. (iPad only)
StartVideoCapture	Starts the video capture.
StopVideoCapture	Stops the video capture.

Public Events

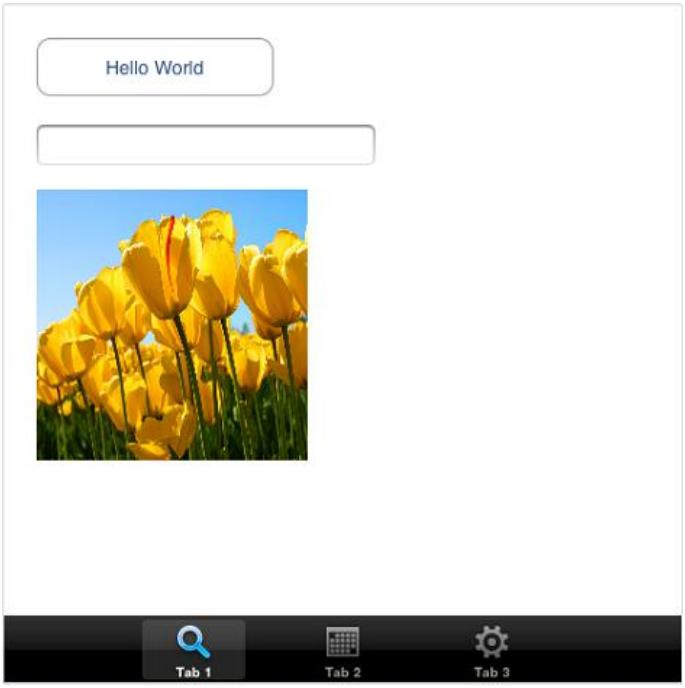
OnDidFinishPickingMediaWithInfo	Access directly to the media dictionary after taking an image or capturing a video with a native reference to an NSDictionary reference.
OnDidFinishPickingImage	Event called when taking an image with native access to a UIImage reference.

Events

OnDidCancel	Event called when cancel is pressed.

OnDidFinishPickingBitmap	Event called when an image is taken or picked and the image is converted to a TBitmap.

TMSFMXNativeUITabBarController



Usage

The TMSFMXNativeUITabBarController displays tabs at the bottom of the window for selecting between the different modes and for displaying the views for that mode.

Published Properties

Badge (TabBarItem level)	A Badge displayed per tab item in the TabBar.
Bitmap (TabBarItem level)	An image display per tab item in the TabBar.
Color (TabBarItem level)	The background color of the tab item in the TabBar.
Enabled (TabBarItem level)	Enables / disables the view of a tab item in the TabBar.
ItemIndex (TabBarItem level)	The current item index of the tab inside the TabBar.
SelectedItem (TabBarController level)	The current selected item in the TabBar.
SelectedItemIndex (TabBarController level)	The current selected item index in the TabBar.
TabEnabled (TabBarItem level)	Enables / disables a tab item in the TabBar.

Text (TabBarItem level)	The text display on a tab in the TabBar.

Public Properties

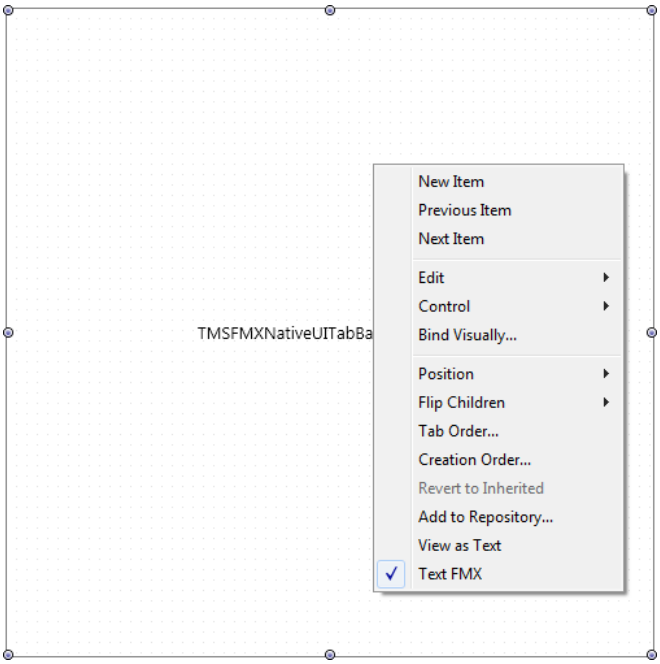
TabBarController	Returns a reference to the native iOS UITabBarController.
ViewController (TabBarItem level)	Returns a reference to the native iOS UIViewController that is used for each tab inside the TabBarController.

Events

OnItemChanged	Event called when a new tab item is selected and the OnItemWillChange event has returned an AllowChange true.
OnItemWillChange	Event called when a different tab will be selected, through this event an AllowChange parameter can be used to optionally disable switching to a different tab item. The AllowChange parameter is true by default.

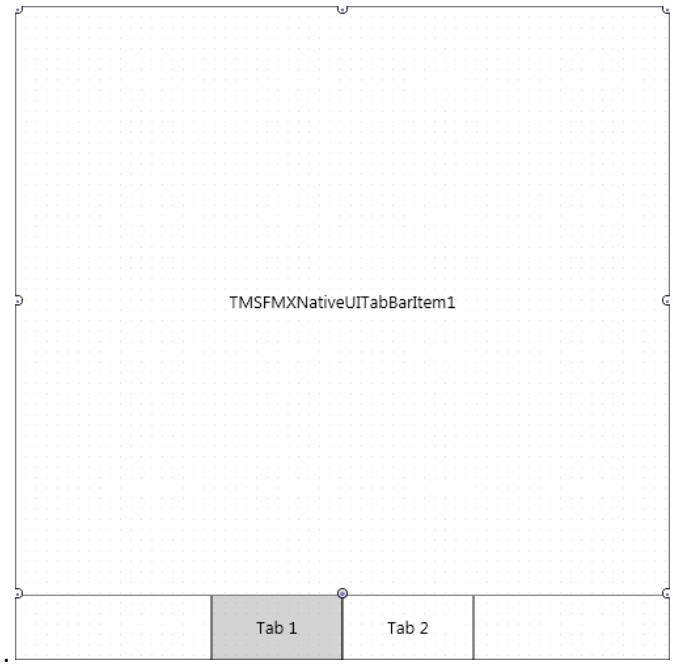
Adding tabs

When dropping a TMSFMXNativeUITabBarController instance on the form (TabBar), it will look similar to any other native iOS control in the TMS iCL package. When right-clicking on the TabBar, you will notice a popup menu with some options to add, remove items and jump to the previous or next page.



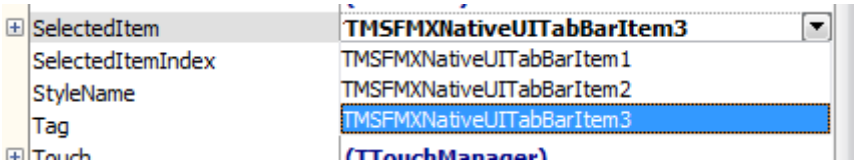
Clicking on New Item, adds a new TMSFMXNativeUITabBarItem instance to the form (TabBarItem), which can only be used inside the TabBar. In the sample below, 2 TabBarItems have been added, and the first TabBarItem is selected.

Clicking on the Tabs below will not change the page, they are currently only an indicator since FireMonkey does not handle designtime messages.



Designtime handling

As mentioned in "Adding tabs", designtime selection and modifications needs to be done through right-clicking the TabBarItem or the TabBar and choosing one of the options. You can also switch TabBarItems by selecting a different item in the SelectedItem property list:



Alternatively you can also set the SelectedItemIndex property that will automatically set the SelectedItem or vice versa.

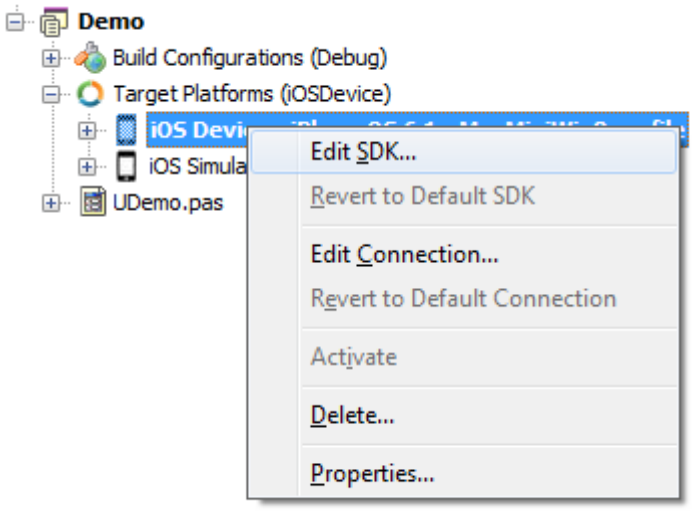
Each TabBarItem inherits from TMSFMXNativeUIView and adds some properties that are used inside a TabBarItem. As it inherits from this class it adds full support to add other TMS FMX Native iOS controls.

Frameworks

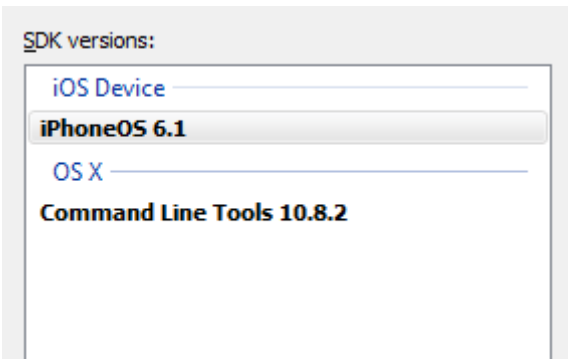
Starting from version 1.2, when deploying to the device, you will encounter linker errors similar to the one below. When adding the SDK through the SDK manager you will notice that it already adds a subset of the available Frameworks in the iOS SDK such as the UIKit and the Foundation framework.

```
[DCC Error] E2597 ld: file not found: /System/Library/Frameworks/ImageIO.framework/ImageIO
[DCC Fatal Error] F2588 Linker error code: 1 ($00000001)
```

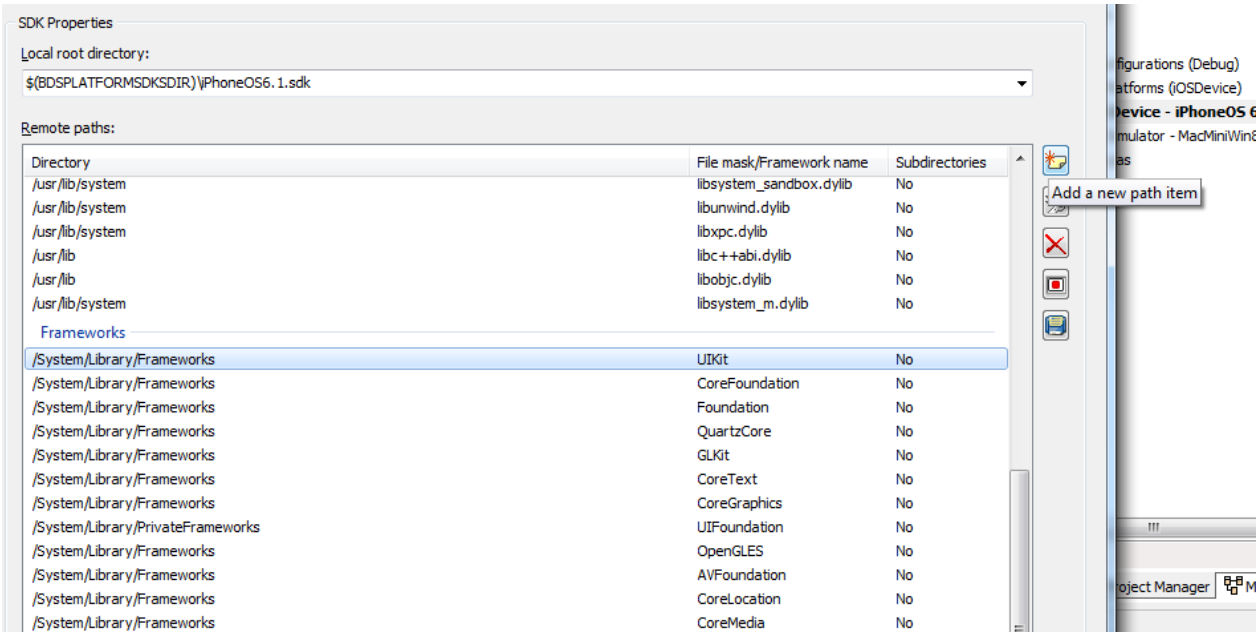
To fix the linker error, you will need to add a reference to the ImageIO framework in the SDK Manager. To add a new framework, right-click on your project and choose the iOS Device target. Right-click on the target and choose "Edit SDK" from the popup menu.



After clicking the correct option in the popup menu, the SDK Manager will popup, showing you which SDK's you have imported and which frameworks are available for each SDK.



Scroll down to the "Frameworks" section for the current SDK you are compiling / linking with. Click inside the "Frameworks" section, for example on the UIKit framework entry, and click on the new button at the top right next to the list to add a new framework entry (path item).

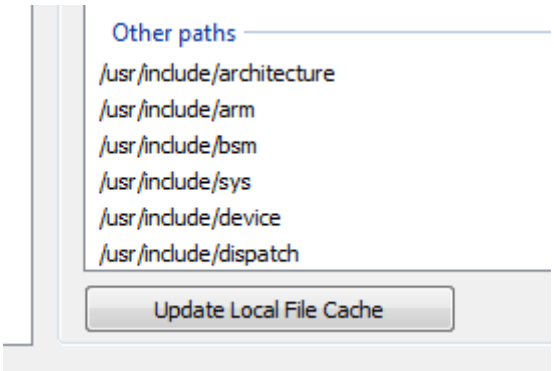


Enter the following information in the popup dialog

Path on remote machine: /System/Library/Frameworks

Framework name: ImageIO

and click on ok. The last step necessary for a correct linking is to update the local SDK directory with the new information. Click on "Update local file cache" at the bottom of the SDK manager:



Targetting for iOS Device should now compile and link without any issues.

It could be possible that, in a next version, the TMS iCL component set requires access to additional frameworks. They will be listed here in the "Frameworks".

View hierarchy

The TMS iCL components can be dropped directly as a child of the main application form, but can also be used as a child of another TMS FMX Native UI control. Included in the set is a TMSFMXNativeUIView control that can be compared with a TPanel in VCL. The view is typically used as a container control that can hold other controls. This is demonstrated below with a small sample.

Drop a TMSFMXNativeUIView on the form and add a TMSFMXNativeUIButton control as child of the view.

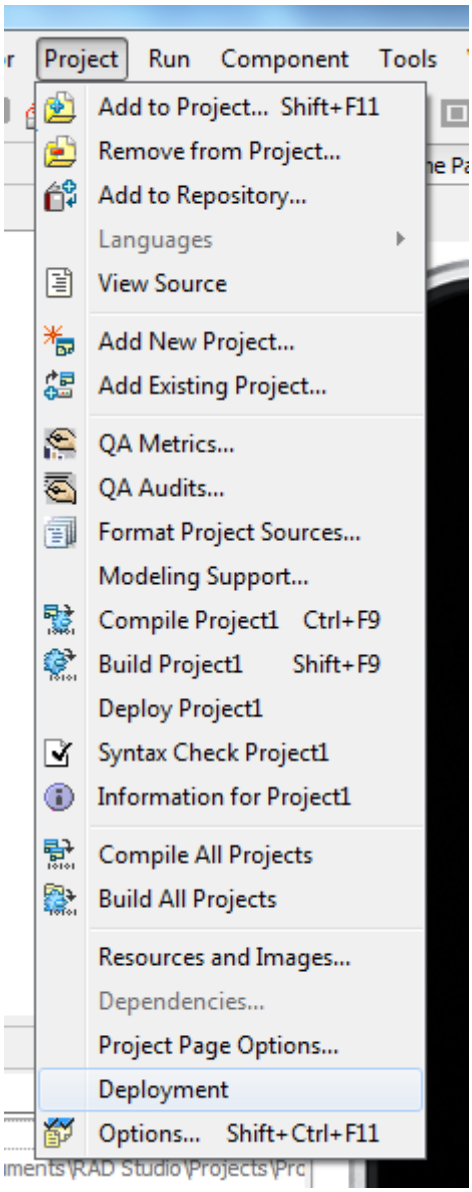
At designtime.

At runtime.

When setting the visible property of the TMSFMXNativeUIView to false, the button will also disappear. If we have a large area of controls and need to apply scrolling, the TMSFMXNativeUIScrollView can be used as a container for other controls. The view can be used as a container control and be linked to a TMSFMXNativeUITableView item's DetailView property and be displayed when clicking on the item.

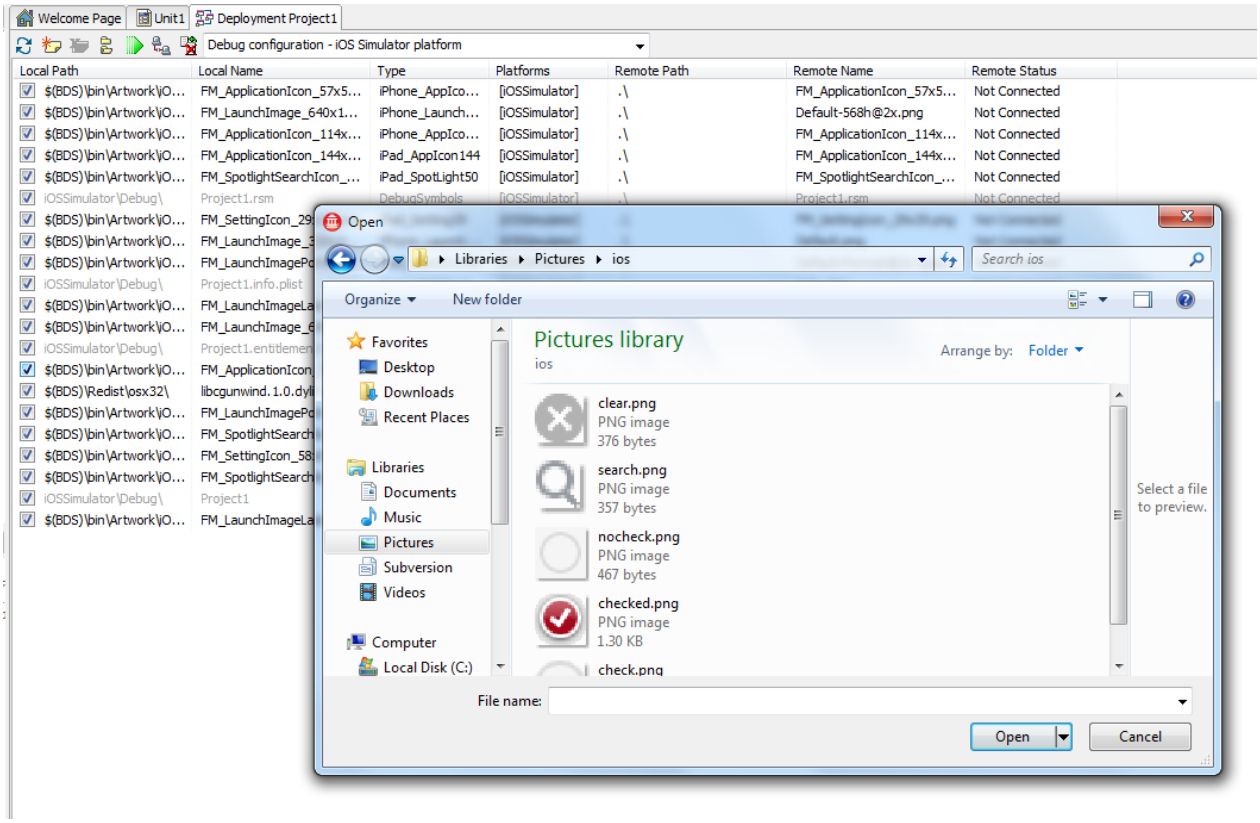
Deployment

At some point your application might have the need to access external files such as images and text files, or perhaps a database that needs to be accessed. When creating a new mobile application, clicking on the project tab and selecting deployment shows a windows where these files can be added.

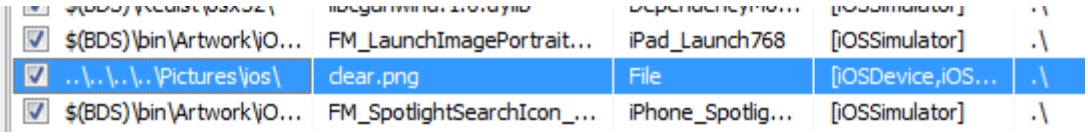


The deployment window already contains files that are deployed along with your application such as the various application icons and launch images.

To add a new file, click on the add button which will popup a file open dialog.



Add the file by clicking open in the dialog. The file will be listed in the deployment window of your project and can be accessed from your application.



To access this file from your application, you need to get the root directory and apply the name of your file as listed in the deployment page. Note that the root directory is read-only, so you will be unable to write data to the file, such as text files. To gain write access to your file you need to copy the file to the documents directory.

Listed below are some helper functions that allow you to access your file and access the root or documents directory.

Root Directory :

```
function GetRootDirectory: String;
begin
  Result := ExtractFilePath(ParamStr(0));
end;
```

Documents Directory (requires **iOSApi.Foundation** and **iOSApi.UIKit** unit):

```
function GetDocumentsDirectory: String;
var
  paths: NSArray;
begin
  Result := '';
  paths :=
TNSArray.Wrap(NSSearchPathForDirectoriesInDomains(NSDocumentDirector
y, NSUserDomainMask, True));
  if paths.count > 0 then
    Result :=
String(TNSString.Wrap(paths.objectAtIndex(0)).UTF8String);
end;
```

Copy a File (requires **iOSApi.Foundation** and **iOSApi.UIKit** units):

```
procedure CopyFile(ASource, ADestination: String);
var
  fileManager: NSFileManager;
  error: NSError;
begin
    fileManager := TNSFileManager.Create;
    fileManager.copyItemAtPath(NSStr(ASource), NSStr(ADestination),
Error);
end;
```

iOS Simulator vs Device

The TMS iCL package supports deployment to simulator and to a real device. The simulator can be helpful in debugging and creating your application without the need to go through the device each time your application is modified. The process of going through the deployment phase is slower when targeting a real device.

We, however, strongly suggest testing your application on a real device at regular intervals during application development, to make sure that the application behaves like it has been developed. There are known limitations and issues in the simulator in terms of look and feel, and it is important

to make sure that these limitations do not occur on a real device since the device will be used in the final stage of development and deployment.

Limitations

Unfortunately, in the current Delphi XE4 release, there is an issue in the RTL. Basically, this RTL issue limits the number of iOS classes that can be simultaneously used (due to a buffer size limitation in RTL).

This means that you will encounter a problem when dropping a high number of different classes of iOS native controls on a FireMonkey form. There is no limitation when using multiple instances of the same class, just on the number of different underlying iOS classes that are imported. There is also no limitation when using many different FireMonkey classes.
We have informed Embarcadero about this problem and it is acknowledged. We hope that Embarcadero will address this limitation as soon as possible in an XE4 update or hotfix.

Resources

TMSFMXNativeUITableViewMail source

```
{*************************************************************************}
{                                                                         }
{ written by TMS Software                                                 }
{            copyright © 2013                                             }
{            Email : info@tmssoftware.com                                 }
{            Web : http://www.tmssoftware.com                             }
{                                                                         }
{ The source code is given as is. The author is not responsible           }
{ for any possible damage done due to the use of this code.               }
{ The complete source code remains property of the author and may         }
{ not be distributed, published, given or sold in any form as such.       }
{ No parts of the source code can be included in any other component      }
{ or application without written authorization of the author.             }
{*************************************************************************}

unit FMX.TMSNativeUITableViewMail;

interface

uses
  Classes, FMX.TMSNativeUITableView, SysUtils
  {$IFDEF IOS}
  ,iOSApi.UIKit, iOSApi.Foundation
  {$ENDIF}
  ;

type
```

```
  TTMSFMXNativeUITableViewMailItem = class(TTMSFMXNativeUITableViewItem)
  private
    FDate: TDateTime;
    FTitle: String;
    FDescription: string;
    FSender: String;
    FUnread: Boolean;
    procedure SetUnread(const Value: Boolean);
  published
    property Sender: String read FSender write FSender;
    property Date: TDateTime read FDate write FDate;
    property Title: String read FTitle write FTitle;
    property Description: string read FDescription write FDescription;
    property Unread: Boolean read FUnread write SetUnread;
  end;

  TTMSFMXNativeUITableViewMailItems = class(TTMSFMXNativeUITableViewItems)
  public
    function CreateItemClass: TCollectionItemClass; override;
  end;

  TTMSFMXNativeUITableViewMailSection =
class(TTMSFMXNativeUITableViewSection)
  public
    function CreateItems: TTMSFMXNativeUITableViewItems; override;
  end;

  TTMSFMXNativeUITableViewMailSections =
class(TTMSFMXNativeUITableViewSections)
  public
    function CreateItemClass: TCollectionItemClass; override;
  end;

  [ComponentPlatformsAttribute(pidiOSSimulator or pidiOSDevice)]
  TTMSFMXNativeUITableViewMail = class(TTMSFMXNativeUITableView)
  public
    constructor Create(AOwner: TComponent); override;
    function CreateSections: TTMSFMXNativeUITableViewSections; override;
    function GetItemHeight(ASection, ARow: Integer): Single; override;
    function GetItemStyle(ASection, ARow: Integer):
TTMSFMXNativeUITableViewItemStyle; override;
    {$IFDEF IOS}
    procedure DoItemCreateCell(Sender: TObject; var ACell: UITableViewCell;
AItemStyle: TTMSFMXNativeUITableViewItemStyle; ASection, ARow: Integer);
override;
```

```
    procedure DoItemCustomizeCell(Sender: TObject; ACell: UITableViewCell;
AItemStyle: TTMSFMXNativeUITableViewItemStyle; ASection, ARow: Integer);
override;
    {$ENDIF}
  end;


implementation

{ TTMSFMXNativeUITableViewMailItems }

function TTMSFMXNativeUITableViewMailItems.CreateItemClass:
TCollectionItemClass;
begin
  Result := TTMSFMXNativeUITableViewMailItem;
end;

{ TTMSFMXNativeUITableViewMailSection }

function TTMSFMXNativeUITableViewMailSection.CreateItems:
TTMSFMXNativeUITableViewItems;
begin
  Result := TTMSFMXNativeUITableViewMailItems.Create(OwnerTableView, Self);
end;

{ TTMSFMXNativeUITableViewMailSections }

function TTMSFMXNativeUITableViewMailSections.CreateItemClass:
TCollectionItemClass;
begin
  Result := TTMSFMXNativeUITableViewMailSection;
end;

{ TTMSFMXNativeUITableViewMail }

constructor TTMSFMXNativeUITableViewMail.Create(AOwner: TComponent);
begin
  inherited;
  if (csDesigning in ComponentState) and not
    ((csReading in Owner.ComponentState) or (csLoading in
Owner.ComponentState)) then
  begin
    Options.Header := 'Mail';
    Options.ToolBar := True;
  end;
end;
```

```
function TTMSFMXNativeUITableViewMail.CreateSections:
TTMSFMXNativeUITableViewSections;
begin
  Result := TTMSFMXNativeUITableViewMailSections.Create(Self);
end;

{$IFDEF IOS}
procedure TTMSFMXNativeUITableViewMail.DoItemCreateCell(Sender: TObject;
  var ACell: UITableViewCell; AItemStyle:
TTMSFMXNativeUITableViewItemStyle;
  ASection, ARow: Integer);
var
  title: UILabel;
  senderName: UILabel;
  description: UILabel;
  date: UILabel;
  r: NSRect;
begin
  r.origin.x := 5;
  r.origin.y := 5;
  r.size.width := ACell.frame.size.width - 100;
  r.size.height := 15;
  senderName :=
TUILabel.Wrap(TUILabel.Wrap(TUILabel.OCClass.alloc).initWithFrame(r));
  senderName.setFont(TUIFont.Wrap(TUIFont.OCClass.systemFontOfSize(12)));
  senderName.setTextColor(TUIColor.Wrap(TUIColor.OCClass.orangeColor));

senderName.setHighlightedTextColor(TUIColor.Wrap(TUIColor.OCClass.whiteColo
r));
  ACell.contentView.addSubview(senderName);

  r.origin.x := Acell.frame.size.width - 100;
  r.origin.y := 5;
  r.size.width := 100;
  r.size.height := 15;
  date :=
TUILabel.Wrap(TUILabel.Wrap(TUILabel.OCClass.alloc).initWithFrame(r));
  date.setFont(TUIFont.Wrap(TUIFont.OCClass.boldSystemFontOfSize(12)));
  date.setTextAlignment(UITextAlignmentRight);
  date.setTextColor(TUIColor.Wrap(TUIColor.OCClass.blueColor));
  date.setHighlightedTextColor(TUIColor.Wrap(TUIColor.OCClass.whiteColor));
  ACell.contentView.addSubview(date);

  r.origin.x := 5;
  r.origin.y := senderName.frame.size.height + senderName.frame.origin.y;
  r.size.width := ACell.frame.size.width - 100;
  r.size.height := 25;
```

```
  title :=
TUILabel.Wrap(TUILabel.Wrap(TUILabel.OCClass.alloc).initWithFrame(r));
  title.setFont(TUIFont.Wrap(TUIFont.OCClass.boldSystemFontOfSize(16)));

title.setHighlightedTextColor(TUIColor.Wrap(TUIColor.OCClass.whiteColor));
  ACell.contentView.addSubview(title);

  r.origin.x := 5;
  r.origin.y := title.frame.origin.y + title.frame.size.height;
  r.size.width := ACell.frame.size.width - 100;
  description :=
TUILabel.Wrap(TUILabel.Wrap(TUILabel.OCClass.alloc).initWithFrame(r));

description.setHighlightedTextColor(TUIColor.Wrap(TUIColor.OCClass.whiteCol
or));
  ACell.contentView.addSubview(description);
end;

procedure TTMSFMXNativeUITableViewMail.DoItemCustomizeCell(Sender: TObject;
  ACell: UITableViewCell; AItemStyle: TTMSFMXNativeUITableViewItemStyle;
  ASection, ARow: Integer);
var
  title: UILabel;
  senderName: UILabel;
  description: UILabel;
  date: UILabel;
  it: TTMSFMXNativeUITableViewItem;
  mailit: TTMSFMXNativeUITableViewMailItem;
  str: NSString;
  r: NSRect;
begin
  senderName := TUILabel.Wrap(ACell.contentView.subviews.objectAtIndex(0));
  date := TUILabel.Wrap(ACell.contentView.subviews.objectAtIndex(1));
  title := TUILabel.Wrap(ACell.contentView.subviews.objectAtIndex(2));
  description :=
TUILabel.Wrap(ACell.contentView.subviews.objectAtIndex(3));

  it := GetItem(ASection, ARow);
  if Assigned(it) and (it is TTMSFMXNativeUITableViewMailItem) then
  begin
    mailit := it as TTMSFMXNativeUITableViewMailItem;
    if mailit.Unread then
    begin
      str := NSStr(ExtractFilePath(ParamStr(0)) + 'unread_mail.png');

ACell.setImage(TUIImage.Wrap(TUIImage.OCClass.imageWithContentsOfFile(str))
);
```

```
    end
    else
      ACell.setImage(nil);

    if Assigned(senderName) then
    begin
      senderName.setText(NSStr(mailit.Sender));
      r := senderName.frame;
      if Assigned(Acell.image) then
        r.origin.x := 15 + Acell.image.size.width
      else
        r.origin.x := 5;
      senderName.setFrame(r);
    end;
    if Assigned(title) then
    begin
      title.setText(NSStr(mailit.Title));
      r := title.frame;
      if Assigned(Acell.image) then
        r.origin.x := 15 + Acell.image.size.width
      else
        r.origin.x := 5;
      title.setFrame(r);
    end;
    if Assigned(description) then
    begin
      description.setText(NSStr(mailit.Description));
      r := description.frame;
      if Assigned(Acell.image) then
        r.origin.x := 15 + Acell.image.size.width
      else
        r.origin.x := 5;
      description.setFrame(r);
    end;
    if Assigned(date) then
    begin
      date.setText(NSStr(DateToStr(mailit.Date)));
    end;
  end;
end;
{$ENDIF}

function TTMSFMXNativeUITableViewMail.GetItemHeight(ASection,
  ARow: Integer): Single;
begin
  Result := 75;
end;
```

```
function TTMSFMXNativeUITableViewMail.GetItemStyle(ASection,
  ARow: Integer): TTMSFMXNativeUITableViewItemStyle;
begin
  Result := isTableViewCellStyleCustom;
end;

{ TTMSFMXNativeUITableViewMailItem }

procedure TTMSFMXNativeUITableViewMailItem.SetUnread(const Value: Boolean);
begin
  FUnread := Value;
  UpdateSectionAtRow(Section.Index, Index);
end;

end.
```