

# TMS iCL

---

Developer guide

# Table of contents

---

1. Getting started	11
1.1 Overview	11
1.1.1 Availability	11
1.1.2 Frameworks	11
1.1.3 View hierarchy	13
1.1.4 Deployment	14
1.1.5 iOS Simulator vs Device	17
1.1.6 Resources	17
2. Reference	22
2.1 TTMSFMXNativeUIButton	22
2.1.1 Usage	22
2.1.2 Published Properties	22
2.1.3 Public Properties	22
2.1.4 Events	22
2.2 TTMSFMXNativeUISearchBar	23
2.2.1 Usage	23
2.2.2 Published Properties	23
2.2.3 Public Properties	23
2.2.4 Events	24
2.3 TTMSFMXNativeUISlider	25
2.3.1 Usage	25
2.3.2 Published Properties	25
2.3.3 Events	25
2.4 TTMSFMXNativeUISwitch	26
2.4.1 Usage	26
2.4.2 Properties	26
2.4.3 Methods	26
2.4.4 Events	26
2.5 TTMSFMXNativeUITableView	27
2.5.1 Overview	27
2.5.2 Properties	48

2.6	TMSFMXNativeUIToolBar	59
2.6.1	Overview	59
2.6.2	Properties	60
2.7	TMSFMXNativeUIPickerView	62
2.7.1	Overview	62
2.7.2	Properties	63
2.8	TMSFMXNativeUIDatePicker	66
2.8.1	Usage	66
2.8.2	Published Properties	66
2.8.3	Public Properties	66
2.8.4	Methods	66
2.8.5	Events	66
2.8.6	Countdown timer	67
2.9	TMSFMXNativeUITextView	68
2.9.1	Overview	68
2.9.2	Properties	69
2.10	TMSFMXNativeUILabel	71
2.10.1	Usage	71
2.10.2	Published Properties	71
2.10.3	Public Properties	71
2.11	TMSFMXNativeUIScrollView	72
2.11.1	Usage	72
2.11.2	Published Properties	72
2.11.3	Public Properties	72
2.11.4	Events	72
2.12	TMSFMXNativeUIProgressView	73
2.12.1	Usage	73
2.12.2	Published Properties	73
2.12.3	Public Properties	73
2.12.4	Events	73
2.13	TMSFMXNativeUISegmentedControl	74
2.13.1	Overview	74
2.13.2	Properties	75
2.14	TMSFMXNativeUIStepper	77
2.14.1	Usage	77

2.14.2	Published Properties	77
2.14.3	Public Properties	77
2.14.4	Events	77
2.15	TMSFMXNativeUITextField	78
2.15.1	Overview	78
2.15.2	Properties	79
2.16	TMSFMXNativeMKMapView	81
2.16.1	Overview	81
2.16.2	Properties	95
2.17	TMSFMXNativeCLGeoCoder	98
2.17.1	Usage	98
2.17.2	Methods	98
2.17.3	Events	98
2.18	TMSFMXNativeFMXWrapper	99
2.18.1	Usage	99
2.18.2	Published Properties	99
2.19	TMSFMXNativeUIImageView	100
2.19.1	Usage	100
2.19.2	Properties	100
2.19.3	Methods	100
2.19.4	Public Properties	101
2.19.5	Face Detection	101
2.20	TMSFMXNativeUIPopoverController	102
2.20.1	Usage	102
2.20.2	Published Properties	102
2.20.3	Public Properties	102
2.20.4	Methods	102
2.21	TMSFMXNativeUIView	103
2.21.1	Usage	103
2.21.2	Published Properties	103
2.21.3	Public Properties	103
2.21.4	Published Events	103
2.22	TMSFMXNativeUIImagePickerController	104
2.22.1	Usage	104
2.22.2	Published Properties	104

2.22.3	Public Properties	104
2.22.4	Methods	105
2.22.5	Public Events	105
2.22.6	Events	105
2.23	TMSFMXNativeUITabBarController	108
2.23.1	Usage	108
2.23.2	Published Properties	108
2.23.3	Public Properties	108
2.23.4	Events	108
2.23.5	Adding tabs	109
2.23.6	Design-time handling	110
2.24	TMSFMXNativeUINavigationController	111
2.24.1	Usage	111
2.24.2	Published Properties	111
2.24.3	Methods	111
2.24.4	Public Properties	111
2.24.5	Published Events	111
2.24.6	Pushing and popping pages (ViewControllers)	111
2.25	TMSFMXNativeUIViewController	113
2.25.1	Usage	113
2.25.2	Published Properties	113
2.25.3	Public Properties	113
2.25.4	Published Events	113
2.26	TMSFMXNativeUIPopoverController	114
2.26.1	Usage	114
2.26.2	Published Properties	114
2.26.3	Public Methods	114
2.27	TMSFMXNativeUIViewController	115
2.27.1	Usage	115
2.27.2	Published Properties	115
2.27.3	Public Methods	115
2.28	TMSFMXNativeUIPageViewController	116
2.28.1	Usage	116
2.28.2	Published Properties	116
2.28.3	Public Properties	116

2.28.4	Public Events	116
2.28.5	Published Events	117
2.29	TMSFMXNativeUIPDFPageViewController	118
2.29.1	Usage	118
2.29.2	Published Properties	118
2.29.3	Public Properties	118
2.29.4	Published Events	118
2.30	TMSFMXNativeUIPDFViewController	119
2.30.1	Usage	119
2.30.2	Published Properties	119
2.30.3	Public Properties	119
2.31	TMSFMXNativeUIActionSheet	120
2.31.1	Usage	120
2.31.2	Published Properties	120
2.31.3	Methods	120
2.31.4	Public functions	121
2.31.5	Public Properties	121
2.31.6	Published Events	121
2.32	TMSFMXNativeMFMailComposeViewController	122
2.32.1	Usage	122
2.32.2	Published Properties	122
2.32.3	Methods	122
2.32.4	Public Properties	122
2.32.5	Published Events	122
2.33	TMSFMXNativeMFMessageComposeViewController	123
2.33.1	Usage	123
2.33.2	Published Properties	123
2.33.3	Public Properties	123
2.33.4	Published Events	123
2.34	TMSFMXNativeUIRichTextView	124
2.34.1	Usage	124
2.34.2	Published Properties	124
2.34.3	Public Properties	124
2.34.4	Public Methods	124
2.34.5	Import and export of (rich) text	128

2.35	TMSFMXNativeUIRichTextViewToolbar	129
2.35.1	Usage	129
2.36	TMSFMXNativeUIFontPicker	130
2.36.1	Usage	130
2.37	TMSFMXNativeUIColorPicker	131
2.37.1	Usage	131
2.38	TMSFMXNativeMPMoviePlayerViewController	132
2.38.1	Usage	132
2.38.2	Published Properties	132
2.38.3	Public Properties	132
2.38.4	Public Methods	133
2.38.5	Published Events	133
2.39	TMSFMXNativeUIActivityViewController	134
2.39.1	Usage	134
2.39.2	Published Properties	134
2.39.3	Public Methods	134
2.40	TMSFMXNativeSLComposeViewController	135
2.40.1	Usage	135
2.40.2	Public Properties	135
2.40.3	Public Methods	135
2.40.4	Published Events	136
2.41	TMSFMXNativeUICollectionView	137
2.41.1	Overview	137
2.41.2	Properties	155
2.42	TMSFMXNativeUIActivityIndicatorView	162
2.42.1	Usage	162
2.42.2	Published Properties	162
2.42.3	Public Methods	162
2.43	TMSFMXNativeUIWebView	163
2.43.1	Usage	163
2.43.2	Published Properties	163
2.43.3	Public Properties	163
2.43.4	Public Methods	163
2.43.5	Published Events	164
2.43.6	Executing Javascript	164

2.43.7	Loading HTML	164
2.44	TMSFMXNativeiCloud	165
2.44.1	Usage	165
2.44.2	Methods	165
2.44.3	Properties	165
2.44.4	Events	166
2.44.5	Supported types	166
2.44.6	Entitlements	166
2.45	TMSFMXNativeiCloudDocument	168
2.45.1	Usage	168
2.45.2	Properties	168
2.45.3	Methods	169
2.45.4	Events	170
2.45.5	Initialization	170
2.45.6	Notes sample	171
2.45.7	Entitlements	171
2.46	TMSFMXNativePDFLib	173
2.46.1	Usage	173
2.46.2	Methods	174
2.46.3	Public Properties	175
2.46.4	Properties	176
2.46.5	Creating a new document	178
2.46.6	Opening an existing document	178
2.46.7	Drawing pages from an existing PDF document	178
2.46.8	Graphics Library	179
2.46.9	Graphics Library Rich Text	179
2.46.10	Text Flow	180
2.46.11	Text Calculation And Overflow	180
2.46.12	Images	180
2.47	TMSFMXNativeMultipeerConnectivity	181
2.47.1	Usage	181
2.47.2	Methods	181
2.47.3	Public Properties	182
2.47.4	Properties	182
2.47.5	Events	183



2.47.6	Managing peers	185
2.47.7	Sending Data	186
2.47.8	Receiving Data	186
2.47.9	Sending and Receiving Files	186
2.48	TMSFMXNativeCLLocationManager	187
2.48.1	Usage	187
2.48.2	Methods	188
2.48.3	Properties	189
2.48.4	Events	189
2.48.5	Sample authorization and managing the location updates	189
2.49	TMSFMXNativeCMMotionManager	191
2.49.1	Usage	191
2.49.2	Methods	192
2.49.3	Properties	193
2.49.4	Events	193
2.49.5	Sample with Device Motion	193
2.50	TMSFMXNativeCMAltimeter	195
2.50.1	Usage	195
2.50.2	Methods	195
2.50.3	Events	195
2.50.4	Sample obtaining relative altitude updates	195
2.51	TMSFMXNativeLocalAuthentication	197
2.51.1	Usage	197
2.51.2	Methods	197
2.51.3	Events	197
2.52	TMSFMXNativeUIDocumentInteractionController	198
2.52.1	Usage	198
2.52.2	Methods	198
2.52.3	Properties	198
2.53	TMSFMXNativeAVPlayerViewController	199
2.53.1	Usage	199
2.53.2	Methods	199
2.53.3	Properties	200
2.53.4	Events	200
2.53.5	Picture in Picture (iOS 9)	201

2.54	TMSFMXNativeCameraViewController	204
2.55	TMSFMXNativeBarCodeScanner	207
2.56	TMSFMXNativeAppShortcuts	208
2.56.1	Overview	208
2.56.2	Properties	209
2.57	TMSFMXNativeSpeechRecognition	215
2.57.1	Usage	215
2.57.2	Published Properties	215
2.57.3	Public Properties	215
2.57.4	Public Methods	216
2.57.5	Published Events	216
2.58	TMSFMXNativeSpeechCommandRecognition	217
2.58.1	Overview	217
2.58.2	Properties	218
2.59	TMSFMXNativeWKWebView	220
2.59.1	Usage	220
2.59.2	Public Properties	220
2.59.3	Public Methods	220
2.59.4	Published Events	221

# 1. Getting started

---

## 1.1 Overview

---

### 1.1.1 Availability

TMS iCL is a set of components for true native iOS application development and is available for **Embarcadero Delphi XE11, C++Builder XE11 or newer releases.**

### 1.1.2 Frameworks

---

Starting from version 1.2, when deploying to the device, you will encounter linker errors like the one below.

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

Here is a list with frameworks that need to be added to the SDK manager in order to have the components build and link correctly. The sample demonstrates how to add the `ImageIO` framework.

#### Framework

AVKit (iOS 8 or later)

CoreMotion

ImageIO

LocalAuthentication (iOS 8 or later)

MapKit

MessageUI

MobileCoreServices

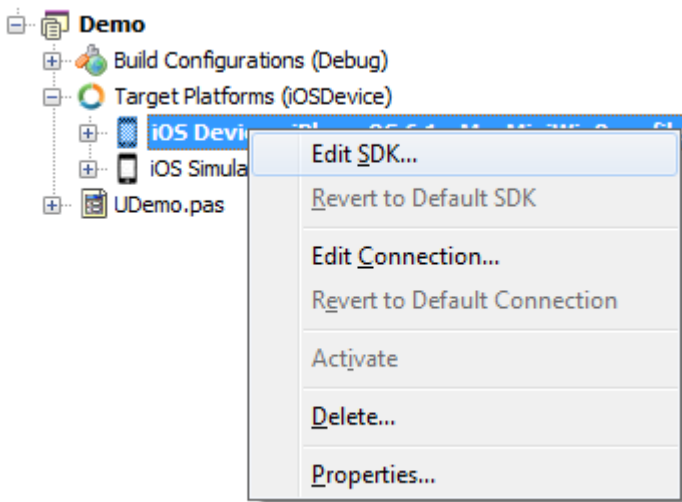
MultipeerConnectivity

Social

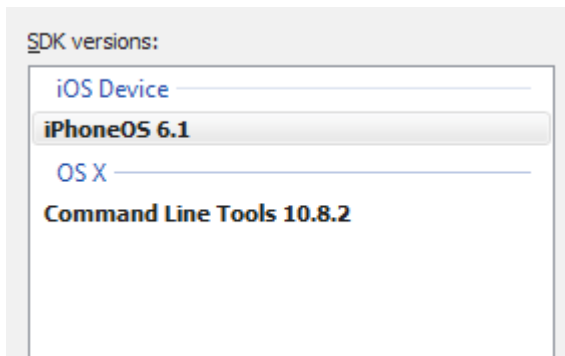
WebKit

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.

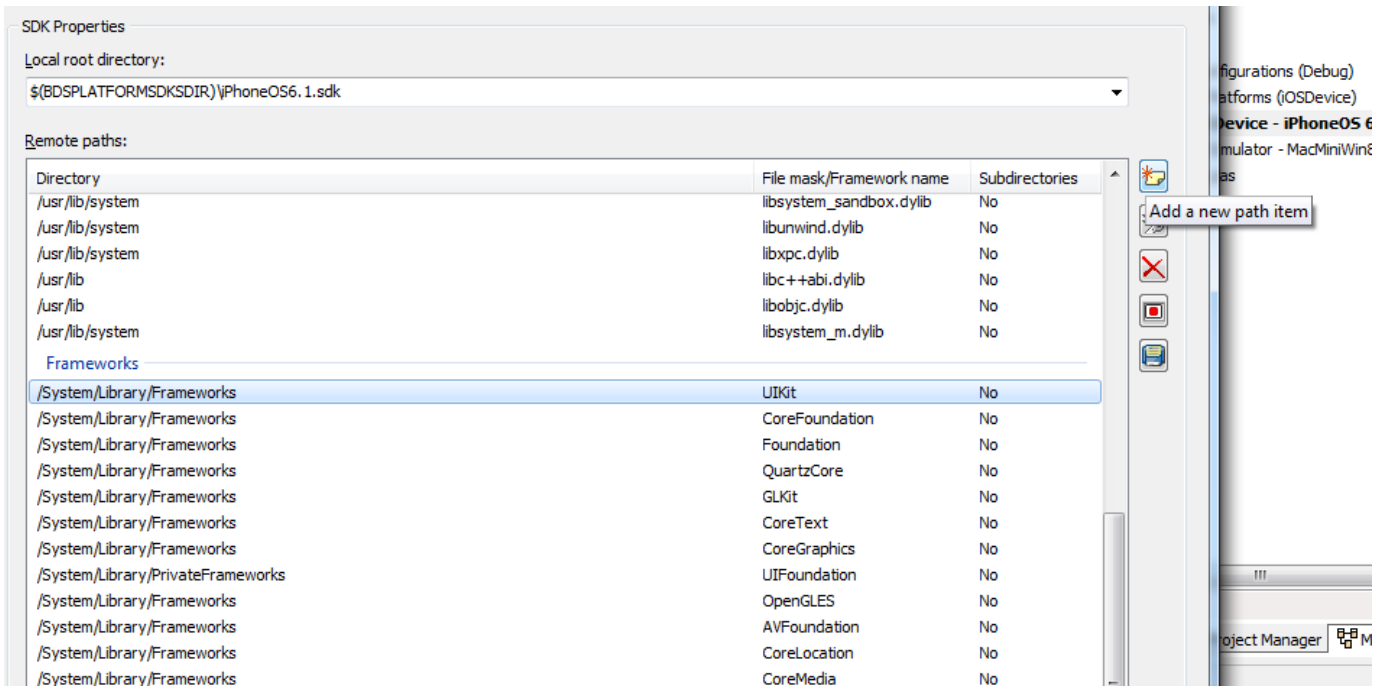
To fix the linker error, you will need to add a reference to the `ImageIO` framework in the SDK Manager. To add a new framework, right-click on your project and choose the iOS Device target. Right-click on the target and choose "Edit SDK" from the popup menu.



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

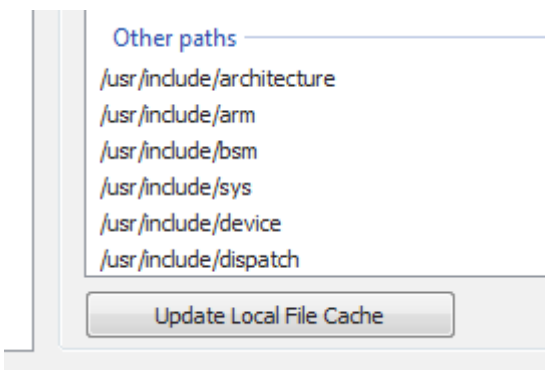
#### Path on remote machine

XE5: `"/System/Library/Frameworks"`

XE6 and later: `"$(SDKROOT)/System/Library/Frameworks"`

**Framework name:** `ImageIO`

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:



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

### 1.1.3 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.**



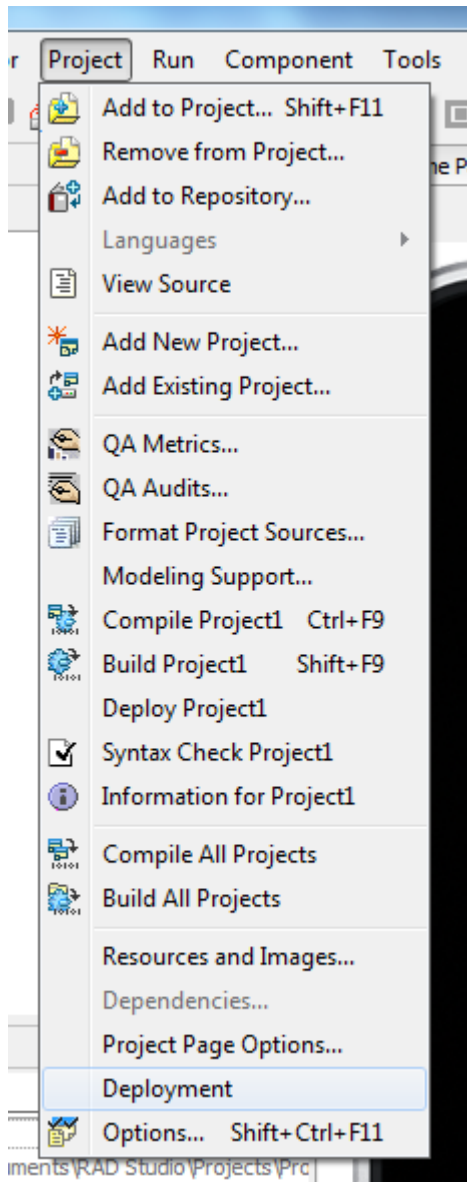
**At runtime.**



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.

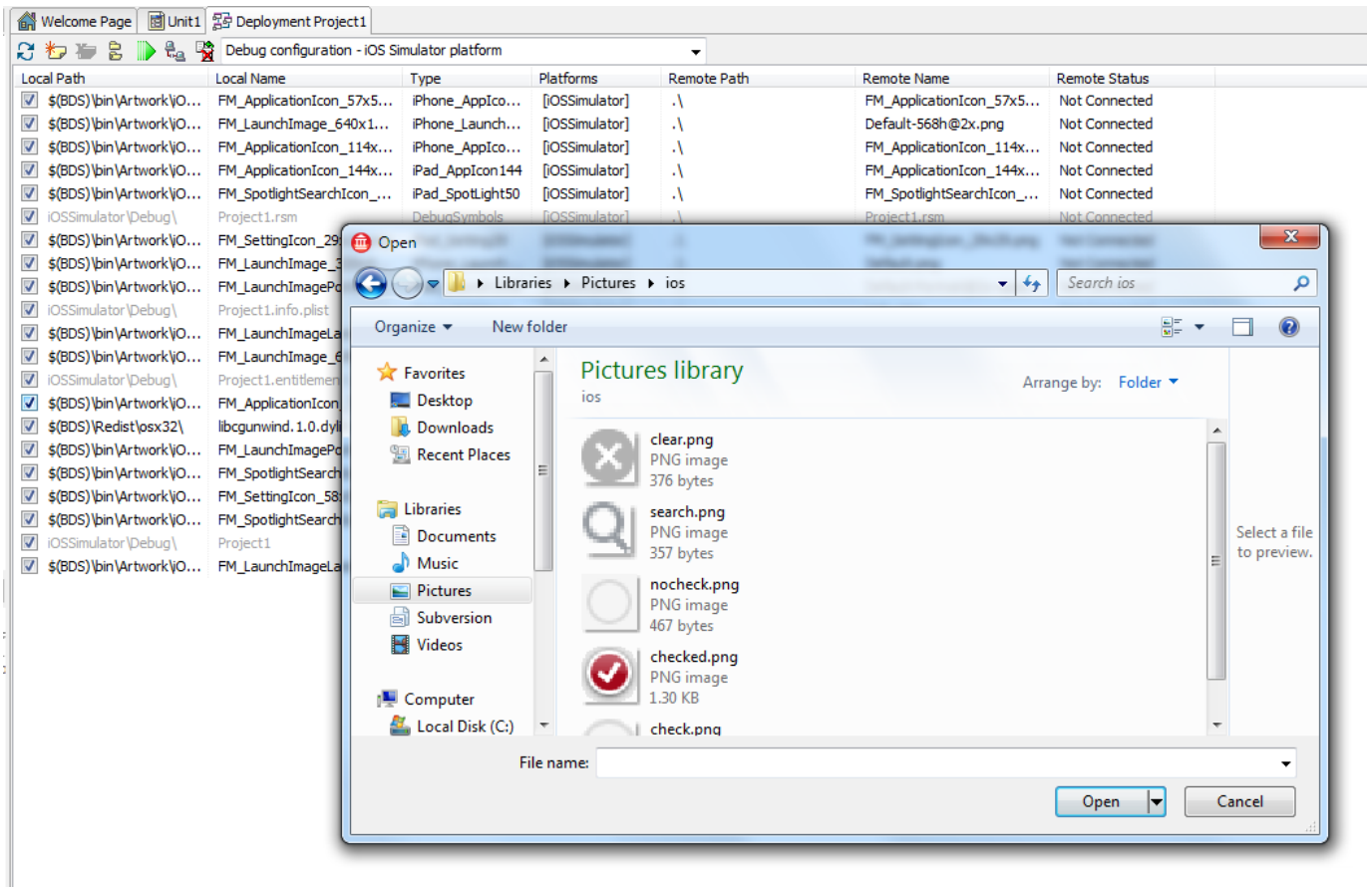
## 1.1.4 Deployment

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.

\$(BDS)\Redist\osx32\	libcgunwind.1.0.dyl...	DebugSymbols...	[iOSSimulator]	.\
✓	\$(BDS)\bin\Artwork\O...	FM_LaunchImagePortrait...	iPad_Launch768	[iOSSimulator]
✓	..\..\..\Pictures\ios\	clear.png	File	[iOSSimulator]
✓	\$(BDS)\bin\Artwork\O...	FM_SpotlightSearchIcon_...	iPhone_Spotlig...	[iOSSimulator]

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(NSDocumentDirectory, 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;
```

## 1.1.5 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.

## 1.1.6 Resources

### TMSFMXNativeUITableViewMail source

```
{*****}
{
{ written by TMS Software }
{ copyright © 2014 }
{ 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 := UILabel.Wrap(UILabel.Wrap(UILabel.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.whiteColor));
    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 := UILabel.Wrap(UILabel.Wrap(UILabel.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 := UILabel.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 := UILabel.Wrap(TUILabel.Wrap(TUILabel.OCClass.alloc).initWithFrame(r));
description.setHighlightedTextColor(TUIColor.Wrap(TUIColor.OCClass.whiteColor));
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 := UILabel.Wrap(ACell.contentView.subviews.objectAtIndex(0));
  date := UILabel.Wrap(ACell.contentView.subviews.objectAtIndex(1));
  title := UILabel.Wrap(ACell.contentView.subviews.objectAtIndex(2));
  description := UILabel.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.

```

## 2. Reference

---

### 2.1 TTMSFMXNativeUIButton

---

#### 2.1.1 Usage

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

#### 2.1.2 Published Properties

Property name	Description
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: <code>bsButtonTypeCustom</code> , <code>bsButtonTypeRoundedRect</code> (Default), <code>bsButtonTypeDetailDisclosure</code> , <code>bsButtonTypeInfoLight</code> , <code>bsButtonTypeInfoDark</code> , <code>bsButtonTypeContactAdd</code>
TextColor	The color of the text of the Button.
TintColor	The color of the text of the Button.
Visible	Shows or hides the Button.

#### 2.1.3 Public Properties

Property name	Description
Button	Returns a reference to the native iOS UIButton.

#### 2.1.4 Events

Event name	Description
OnClick	Event called when clicking on the Button.

## 2.2 TTMSFMXNativeUISearchBar

---

### 2.2.1 Usage

The `TTMSFMXNativeUISearchBar` 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.

### 2.2.2 Published Properties

Property name	Description
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.

### 2.2.3 Public Properties

Property name	Description
SearchBar	Returns a reference to the native iOS UISearchBar.

## 2.2.4 Events

---

<b>Event name</b>	<b>Description</b>
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.

---



## 2.3 TTMSFMXNativeUISlider

---

### 2.3.1 Usage

A `TTMSFMXNativeUISlider` 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.

### 2.3.2 Published Properties

Property name	Description
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.

Property name	Description
Slider	Returns a reference to the native iOS UISlider.

### 2.3.3 Events

Event name	Description
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.

## 2.4 TTMSFMXNativeUISwitch

---

### 2.4.1 Usage

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

### 2.4.2 Properties

---

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

---

### 2.4.3 Methods

---

Method name	Description
Switch	Returns a reference to the native iOS UISwitch.

---

### 2.4.4 Events

---

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

---

## 2.5 TTMSFMXNativeUITableView

---

### 2.5.1 Overview

#### Usage

An instance of `TTMSFMXNativeUITableView` 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.

#### Methods

Method name	Description
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 <code>Options.Editing.Enabled</code> .
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

Event name	Description
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 <code>BeginRefreshing</code> .
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 <code>EndRefreshing</code> .
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 <code>AccessoryView</code> 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 <code>Bitmap</code> for an item.
OnGetItemDescription	Event called to return a <code>Description</code> for an item.
OnGetItemDetailView	Event called to return a <code>DetailView</code> for an item. The <code>DetailView</code> 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 <code>AccessoryView</code> 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 <code>SubDetailView</code> for an item. The <code>SubDetailView</code> can be linked to another TMS FMX Native UI Control.
OnGetItemText	Event called to return the text of an item.

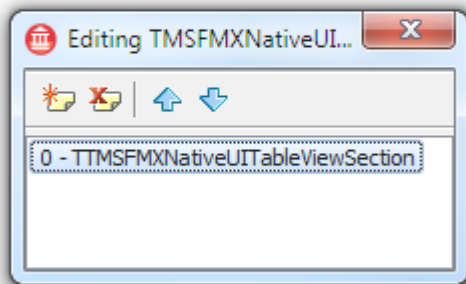
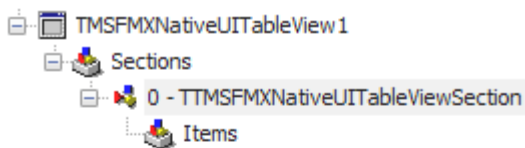
Event name	Description
OnGetNumberOfRowsInSection	Event called that specifies the number of rows in a section.
OnGetNumberOfSections	Event called that specifies the number of sections in a TableView.
OnGetSectionForSectionIndexTitle	Event called that returns the section for a specific index title. The 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 <code>atTableViewCellAccessoryDetailDisclosureButton</code> .
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

Event name	Description
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 <code>OnItemCustomizeCell</code> 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”, ...

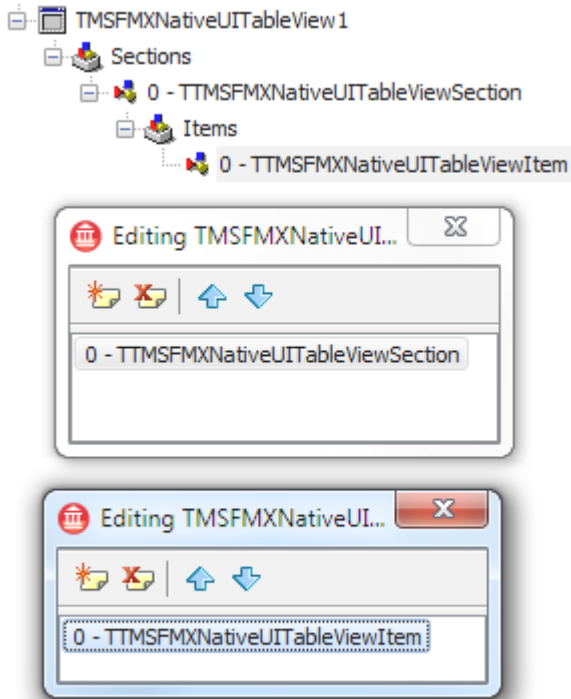
Sections can also be added programmatically:

```
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';
```



Cars
<b>Mercedes</b>
<b>Audi</b>
<b>BMW</b>
Nature
<b>Birds</b>
<b>Plants</b>
Sports
<b>Soccer</b>
<b>Baseball</b>

## 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';
```

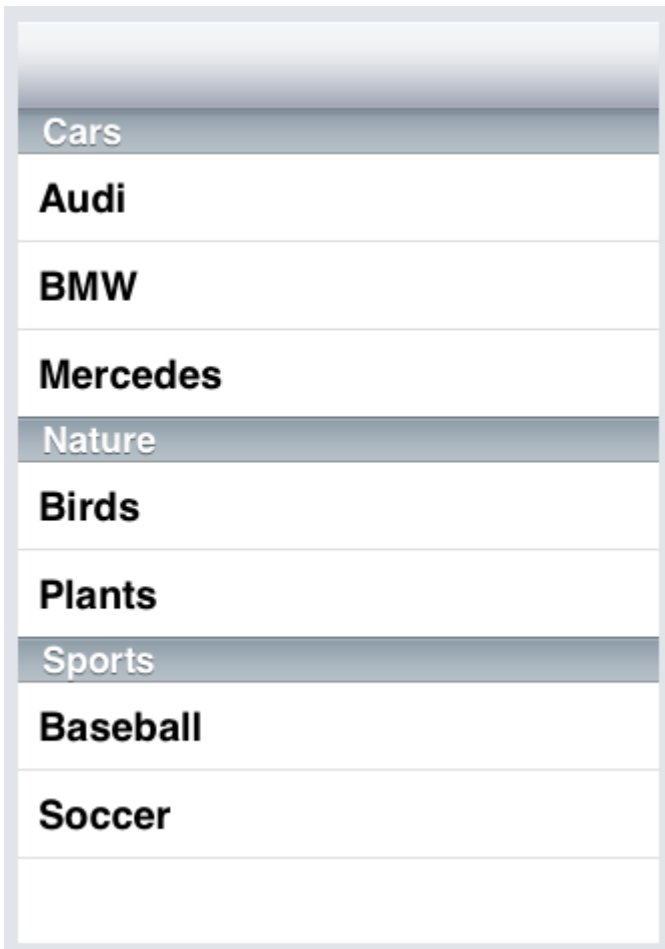
```
s.Items.Add.Text := 'Baseball';  
s.Items.Sort;
```

<b>Cars</b>
<b>Audi</b>
<b>BMW</b>
<b>Mercedes</b>
<b>Nature</b>
<b>Birds</b>
<b>Plants</b>
<b>Sports</b>
<b>Baseball</b>
<b>Soccer</b>

### 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;
```



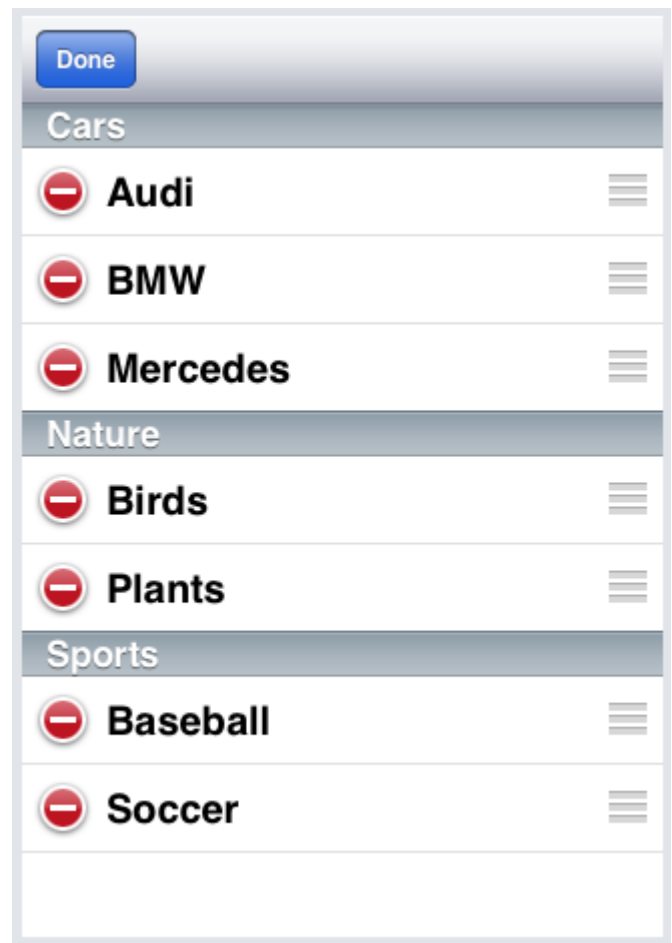
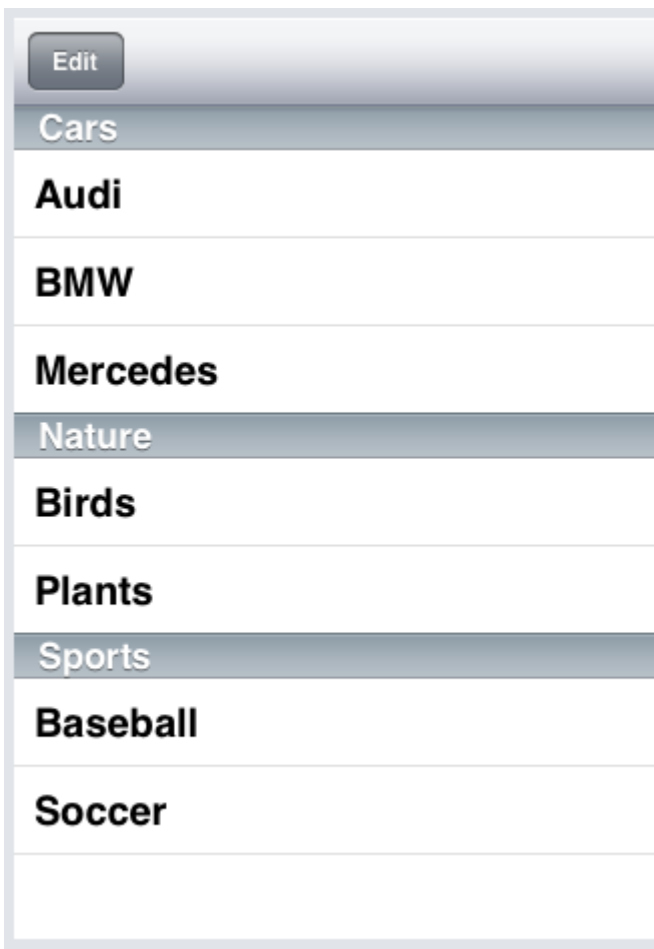
Cars
Audi
BMW
Mercedes
Nature
Birds
Plants
Sports
Baseball
Soccer

## 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 mode there 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 `UITableViewCellEditingStyleDelete` by default. The `EditStyle` can be set to `UITableViewCellEditingStyleInsert` to show a plus button, or set to `UITableViewCellEditingStyleNone` 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: TMSFMXNativeUITableSection;
begin
  TMSFMXNativeUITable1.Options.ToolBar := True;
  TMSFMXNativeUITable1.Options.Editing.Enabled := True;

  s := TMSFMXNativeUITable1.Sections.Add;
  s.Header := 'Cars';
  s.Items.Add.Text := 'Mercedes';
  s.Items.Add.Text := 'Audi';
  s.Items.Add.Text := 'BMW';
```

```

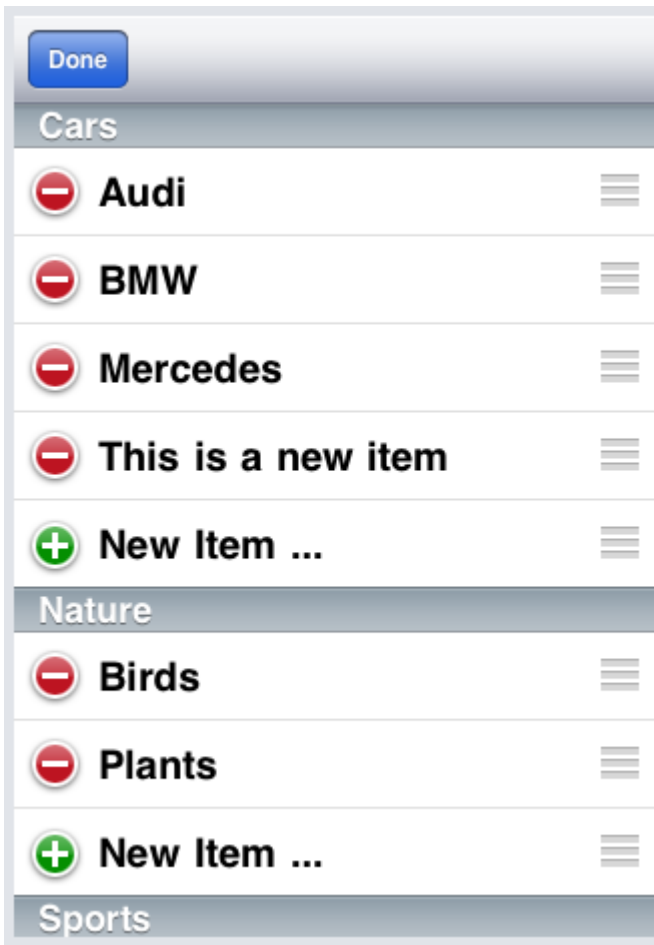
s.Items.Sort;
with s.Items.Add do
begin
  Text := 'New Item ...';
  EditStyle := esTableViewCellEditingStyleInsert;
end;

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: TMSFMXNativeUITableViewItem;
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: TMSFMXNativeUITableViewSection;
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';
```

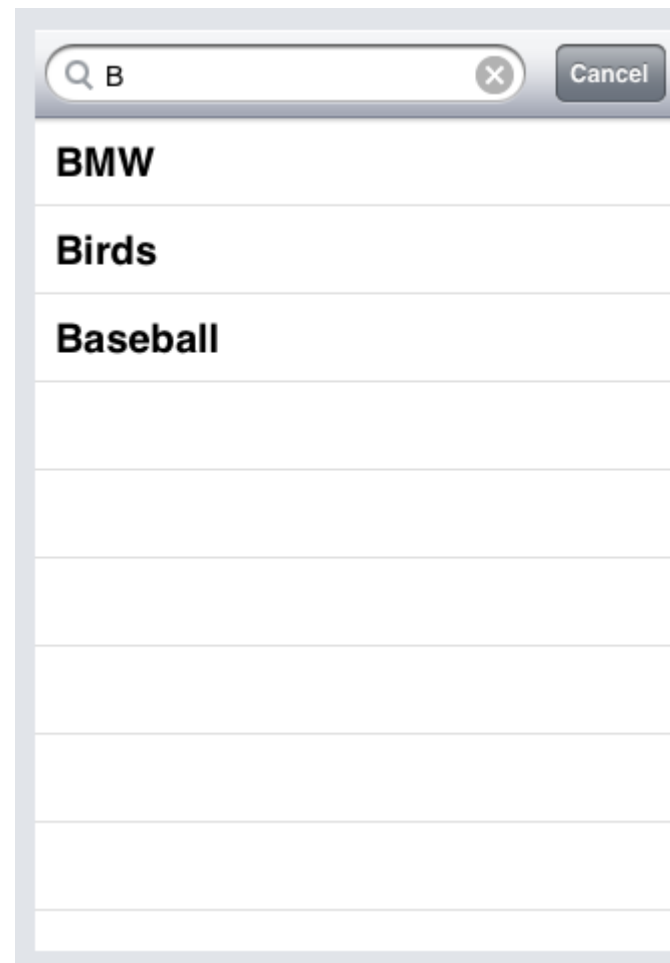
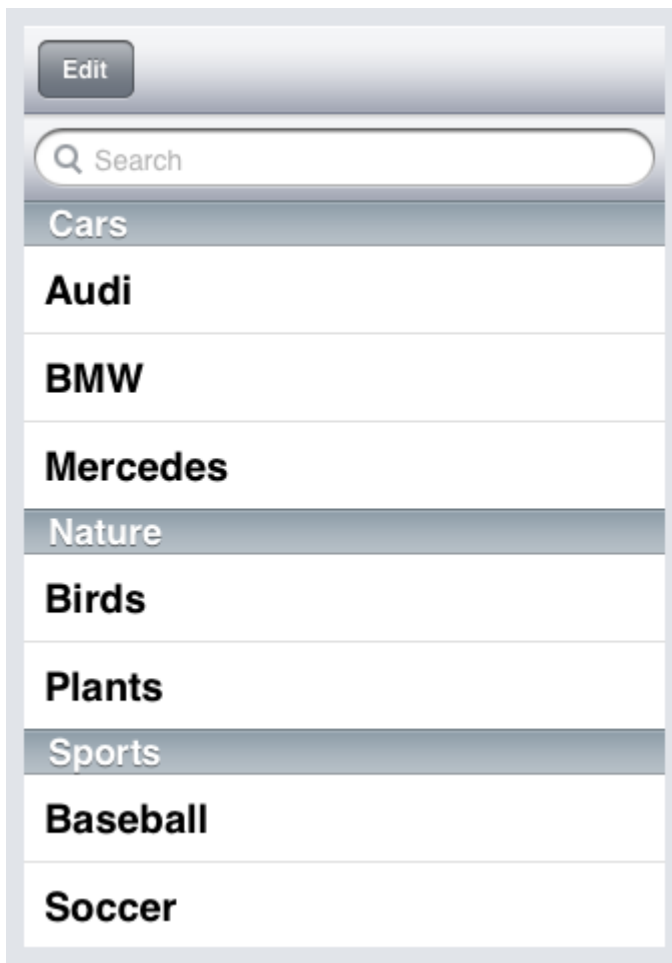
```

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;

```



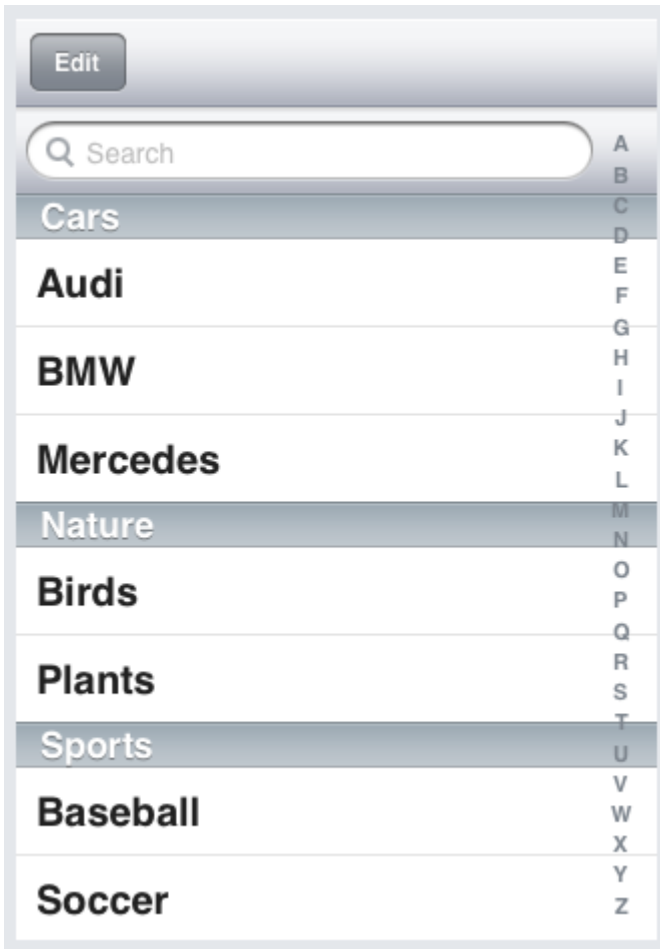
More information on events is explained in the [events table](#) overview.

## 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.



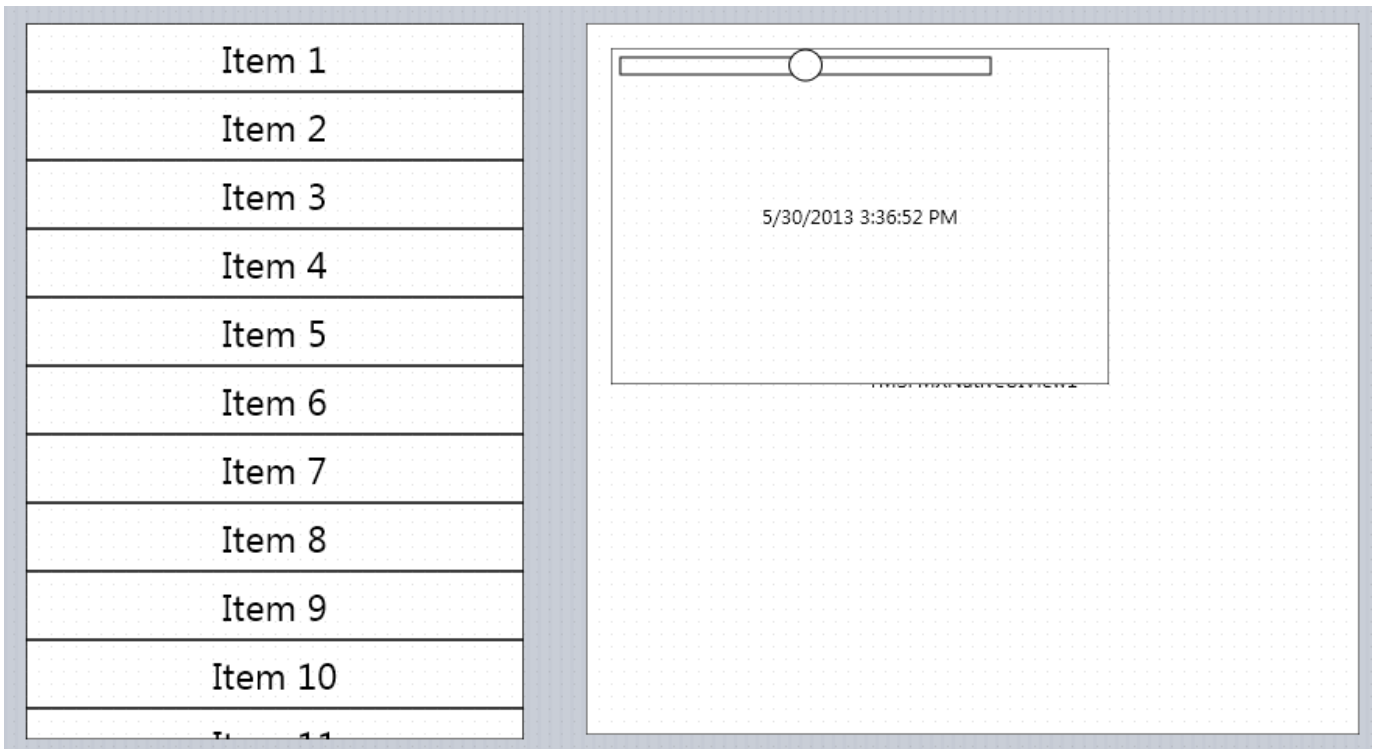
### DetailView and SubDetailView

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.

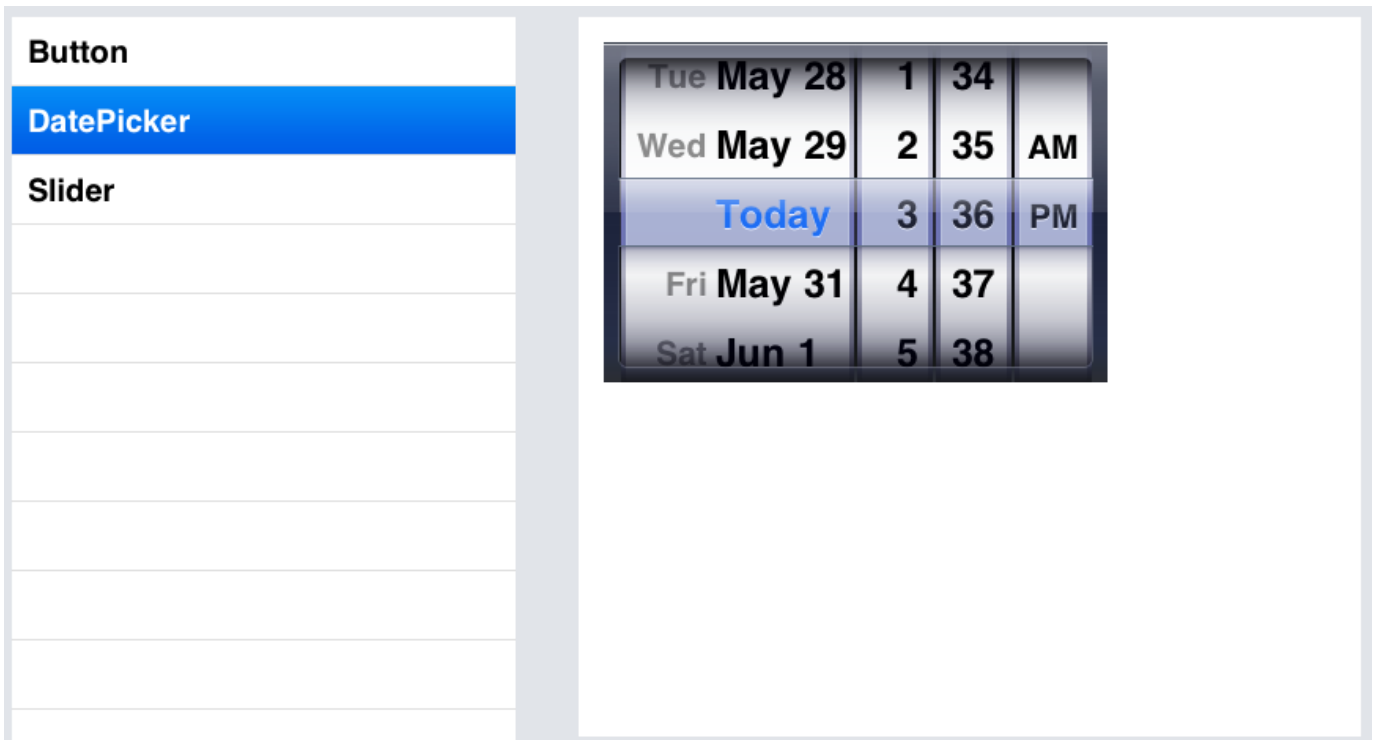
```

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;
end;

```

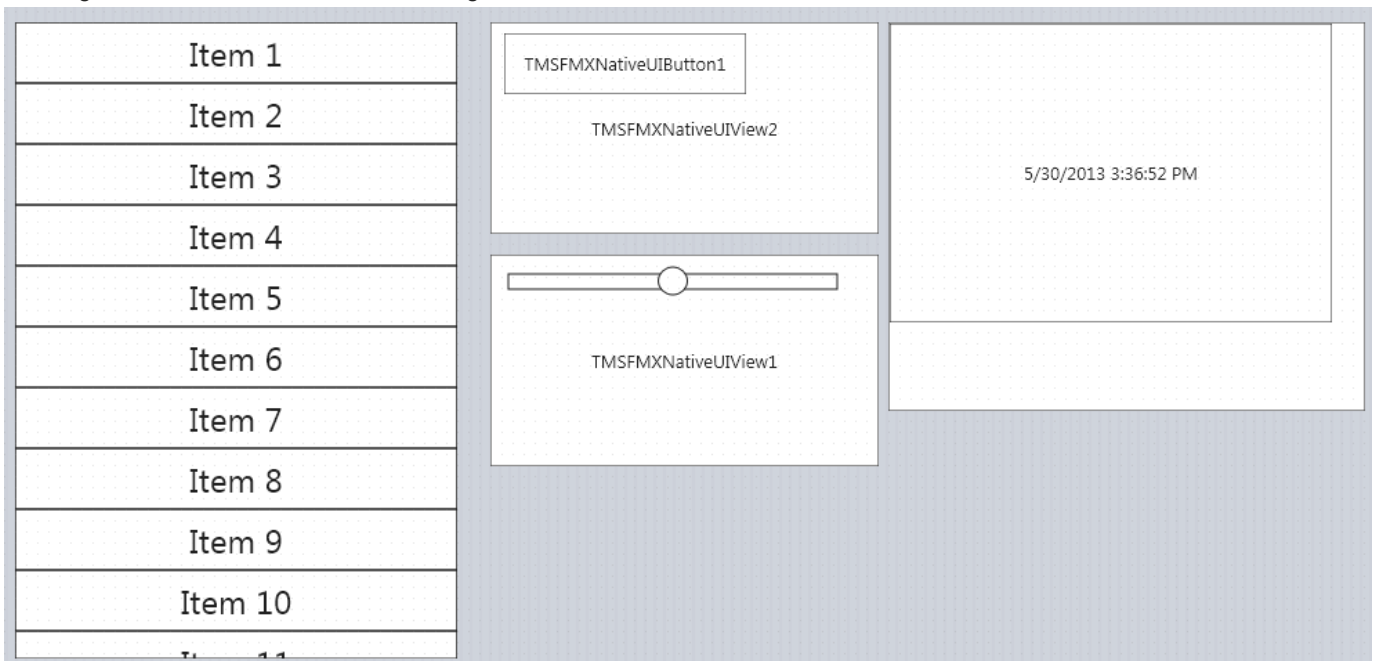
The container view is the `TMSFMXNativeUIView1` and is linked to the `DetailView` property of the `TMSFMXNativeUITableView1`. The 3 children of the view each need to be assigned to the item's `DetailView` property and need to be set `Visible false`.

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:

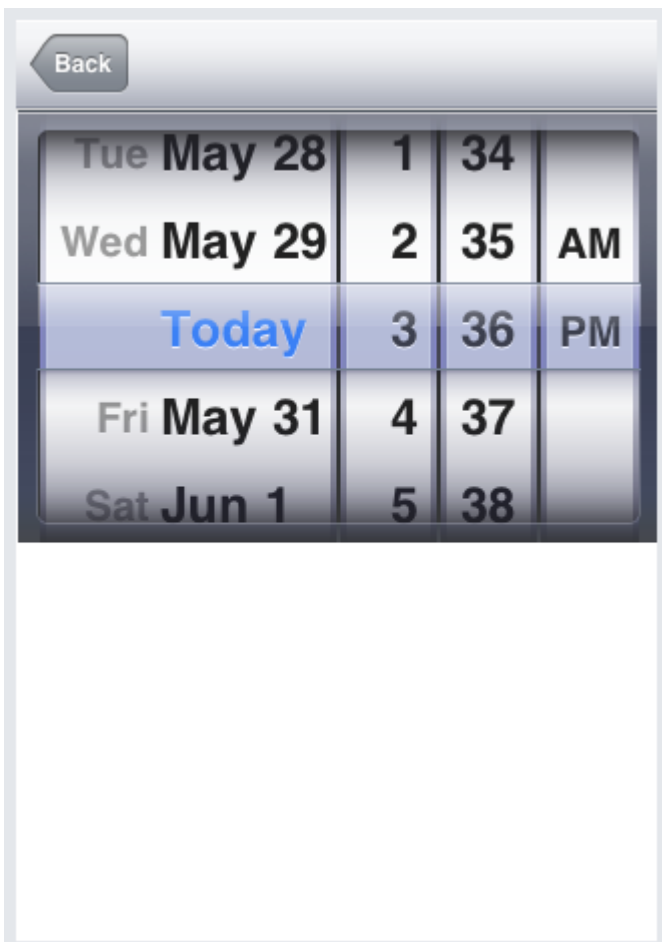
```
TMSFMXNativeUIDatePicker1.Visible := False;
TMSFMXNativeUIView1.Visible := False;
TMSFMXNativeUIView2.Visible := False;
```

```

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;
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 each other 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 each other 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.

Item 1	Item 1	Item 1
Item 2	Item 2	Item 2
Item 3	Item 3	Item 3
Item 4	Item 4	Item 4
Item 5	Item 5	Item 5
Item 6	Item 6	Item 6
Item 7	Item 7	Item 7
Item 8	Item 8	Item 8
Item 9	Item 9	Item 9
Item 10	Item 10	Item 10
Item 11	Item 11	Item 11

The code that accompanies this sample is shown below.

```
var
  s: TMSFMXNativeUITableViewSection;
  it: TMSFMXNativeUITableViewItem;
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;
```

```

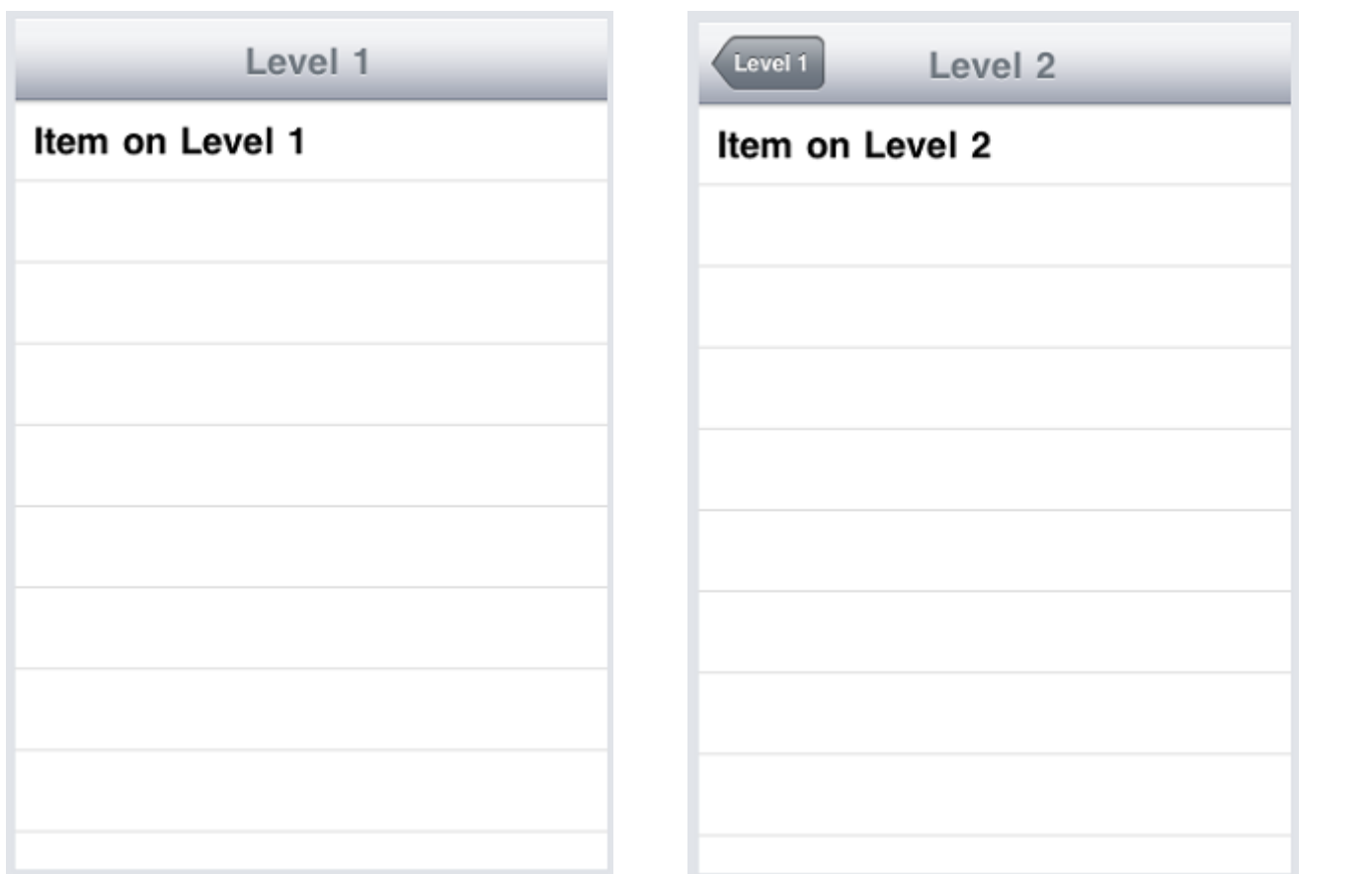
it := s.Items.Add;
it.Text := 'Item on Level 2';
it.SubDetailView := TMSFMXNativeUITableView3;

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;

```

Section 0
<b>S 0 Item 0</b>
<b>S 0 Item 1</b>
<b>S 0 Item 2</b>
Section 1
<b>S 1 Item 0</b>
<b>S 1 Item 1</b>
<b>S 1 Item 2</b>
<b>S 1 Item 3</b>
<b>S 1 Item 4</b>

## 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 `UITableView` 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 `UITableView` item and display custom data.

The Source of the `TMSFMXNativeUITableView` can be found below in the [Resources](#) chapter

## Custom Items

The `UITableView` 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 `UITableView` exposes 2 additional public procedures and events that can be implemented to customize your `UITableView` 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 customcollection. 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) you will notice that custom label instances are added and customized linking to the custom collection.

## 2.5.2 Properties

### Overview

#### PUBLISHED PROPERTIES

Property name	Description
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.
<a href="#">Options</a>	A set of configurable options for the TableView
<a href="#">Sections[Index]</a>	A collection of Sections used in the TableView.
Visible	Shows / Hides the TableView.

#### PUBLIC PROPERTIES

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



## Sections

### OVERVIEW

Property name	Description
Header	The header of the section
Footer	The footer of the section.
<a href="#">Items[Index]</a>	Collection of items within a section
LookUpID	The LookUpID of the section that is used in combination with custom lookup items.

[Go back to Properties](#)

ITEMS

Property name	Description
AccessoryType	The type of standard accessory view the item should use (normal state). Standard accessory types are: <code>atTableViewCellAccessoryNone</code> <code>atTableViewCellAccessoryDisclosureIndicator</code> <code>atTableViewCellAccessoryDetailDisclosureButton</code> <code>atTableViewCellAccessoryCheckmark</code>
AccessoryView	A view that is used, typically as a control, on the right side of the item (normal state).
Bitmap	Sets the image of an item in the TableView.
BitmapFile	A direct link to an image file located in the root or documents directory.
BitmapLink	A link to another <code>TBitmap</code> instance which can be used multiple times to save resources.
BitmapSize	The size of an image used to resize. The <code>BitmapSize</code> is <code>-1</code> by default which will load the original size of the image. The image is resized with aspect ratio.
CanMove	Sets whether an item can be moved to another location in the TableView or not.
Enabled	Enables / disables the item.
Description	The description of the item.
DetailView	A <code>DetailView</code> that is used when the user selects an item. The <code>DetailView</code> is pushed in the View linked to the TableView's <code>DetailView</code> property. The <code>DetailView</code> on item level and on TableView can be linked to other types of TMS FMX Native UI controls.
EditingAccessoryType	The type of standard accessory view the item should use (editing state). Standard accessory types are: <code>atTableViewCellAccessoryNone</code> <code>atTableViewCellAccessoryDisclosureIndicator</code> <code>atTableViewCellAccessoryDetailDisclosureButton</code> <code>atTableViewCellAccessoryCheckmark</code>
EditingAccessoryView	A view that is used, typically as a control, on the right side of the item (editing state).
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.
Height	The height of an item in the TableView when a value different from <code>-1</code> is specified. If the Value is <code>-1</code> the <code>RowHeight</code> property on Options level is used.
ShowEditMenu	Shows a "Copy" edit menu when a tap and hold operation occurs on an item.
Style	The Style of an item. Various styles can be applied <code>UITableViewCellStyleDefault</code> A simple style for an item with a text label (black and left-aligned) and an optional image view. <code>UITableViewCellStyleValue1</code> A style for an item with a label on the left side of the item with left-aligned and black text;

Property name	Description
	<p>on the right side is a label that has smaller blue text and is right-aligned.</p> <p><code>UITableViewCellStyleValue2</code></p> <p>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.</p>
	<p><code>UITableViewCellStyleSubtitle</code></p> <p>A style for an item with a left-aligned label across the top and a left-aligned label below it in smaller gray text.</p>
SubDetailView	<p>A SubDetailView that is used when the user selects an item. The SubDetailView is pushed in place of the main TableView. The <code>SubDetailView</code> on item level can be linked to other types of TMS FMX Native UI controls.</p>
Text	<p>The text of the item.</p>

[Go back to Sections](#)

## Options

### OVERVIEW

Property name	Description
AllowsMultipleSelection	A Boolean value that determines whether users can select more than one row outside of editing mode or not.
AllowsMultipleSelectionDuringEditing	A Boolean value that controls whether users can select more than one item simultaneously in editing mode or not.
AllowsSelection	A Boolean value that determines whether users can select a row or not.
AllowsSelectionDuringEditing	A Boolean value that determines whether users can select items while the receiver is in editing mode or not.
<a href="#">Editing</a>	A set of configurable editing options for the TableView.
Header	The header of the TableView displayed in the toolbar.
Layout	The Layout of the TableView. The TableView can have 2 layout modes: plain and grouped layout mode.
<a href="#">LookUp</a>	A set of configurable lookup options for the TableView.
<a href="#">Refreshing</a>	A set of configurable refreshing options for the TableView.
RowHeight	The default rowheight of the TableView items. The rowheight can be configured per item with the <a href="#">Height</a> property.
<a href="#">Scrolling</a>	A set of configurable scrolling options for the TableView.
<a href="#">Searching</a>	A set of Configurable searching options for the Tableview.
SeparatorColor	The color of separator items in the table view.
SeparatorStyle	The style for items used as separators.
ToolBar	Shows / hides the toolbar on the TableView.

[Go back to Properties](#)

## EDITING

Property name	Description
EditButton	Shows an edit button in the toolbar which toggles the TableView from normal to edit mode or vice versa. ( <a href="#">Options.Toolbar</a> and <a href="#">Options.Editing.Enabled</a> properties need to be true)
Enabled	Enables or disables editing capabilities in the TableView.

[Go back to Options](#)

## LOOKUP

Property name	Description
Items	A collection of custom lookup items.
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 <code>Items</code> collection under the Lookup property. The linked can be done by specifying an ID on the item and a <code>LookUpID</code> on the section.

[Go back to Options](#)

## REFRESHING

Property name	Description
AutoEnd	Automatically ends refresh operation after <code>BeginRefreshing</code> is called or a swipe down operation is performed.
Enabled	Enables Refreshing, disabled by default.
TintColor	Sets the tint-color of the Refresh indicator on the TableView

[Go back to Options](#)



## SCROLLING

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

[Go back to Options](#)

## SEARCHING

Property name	Description
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.
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.
ScrollPosition	The position in the TableView (top, middle, bottom) to which a given row is scrolled when using searching mode.

[Go back to Options](#)

## 2.6 TTMSFMXNativeUIToolBar

---

### 2.6.1 Overview

---

#### Usage

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

#### Methods

Method name	Description
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(AltItem: UIBarButtonItem): TTMSFMXNativeUIToolBarItem;	Returns the item by passing a reference to the native <code>UIBarButtonItem</code> that is linked to an item.

#### Events

Event name	Description
OnItemClick	Event called when clicking on an item (button)

---

## 2.6.2 Properties

---

### Overview

#### PUBLISHED PROPERTIES

Property name	Description
<a href="#">Items[Index]</a>	The items displayed on the ToolBar.
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

Property name	Description
ToolBar	Returns a reference to the native iOS <code>UIToolbar</code> .

## Items

Property name	Description
Action	Property to assign an action combined with an action list.
Bitmap	The bitmap used inside an item.
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 <code>Kind</code> property is set to <code>ikCustom</code> .
Enabled	Enables / disables an item.
Kind	The kind of item that is display in the ToolBar, an item can be a normal, system or custom item.
Style	The style of an item, applied when using normal or system Kind. The style can be plain, done or bordered.
SystemItem	When setting the kind to <code>ikSystem</code> , the <code>SystemItem</code> property determines which icon is display inside the button.
Text	The text of a button.
Visible	Shows / hides a button.

[Go back to Properties](#)

## 2.7 TMSFMXNativeUIPickerView

---

### 2.7.1 Overview

#### 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.

#### Methods

Method name	Description
<code>SelectRowInColumn(ARow, AColumn: Integer; AAnimated: Boolean);</code>	Selects a specific row (item) in a specific column.
<code>SelectedRowForColumn(AColumn: Integer): Integer;</code>	Returns the selected index of a row (item) in a specific column.

#### Events

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

## 2.7.2 Properties

---

### Overview

#### PUBLISHED PROPERTIES

Property name	Description
<a href="#">Columns</a>	Collection of columns used in the pickerView.
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

Property name	Description
PickerView	Returns a reference to the native iOS <code>UIPickerView</code> .

---

## Columns

### OVERVIEW

Property name	Description
<a href="#">Items</a>	Collection of items per column.

---

[Go back to Properties](#)



## ITEMS

Property name	Description
Text	The text of an item per column.

---

[Go back to Columns](#)

## 2.8 TMSFMXNativeUIDatePicker

---

### 2.8.1 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.

### 2.8.2 Published Properties

Property name	Description
CountDownDuration	This property (in seconds) is only used to display a number when the <code>Mode</code> is set to <code>dpmDatePickerModeCountDownTimer</code> . 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 <code>dpmDatePickerModeDateAndTime</code> .
Visible	Shows / hides the DatePicker.

### 2.8.3 Public Properties

Property name	Description
DatePicker	Returns a reference to the native iOS <code>UIDatePicker</code> .

### 2.8.4 Methods

Methods name	Description
<code>NSDateToDate(NSDate): TDateTime;</code>	Returns a <code>TDateTime</code> from a native iOS <code>NSDate</code> instance.

### 2.8.5 Events

Events name	Description
OnValueChanged	Event called when a date / value has changed.

## 2.8.6 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 := dpmDatePickerModeCountDownTimer1;
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;
```

## 2.9 TMSFMXNativeUITextView

---

### 2.9.1 Overview

#### 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.

#### Events

Event name	Description
<code>OnChanged</code>	Event called when the text of the TextField has changed.
<code>OnDidBeginEditing</code>	Event called when editing did begin.
<code>OnDidChangeSelection</code>	Event called when selection of the text did change.
<code>OnDidEndEditing</code>	Event called when editing did end.
<code>OnShouldBeginEditing</code>	Event called when editing should begin.
<code>OnShouldChangeTextInRange</code>	Event called when a specified text in range should be changed. The <code>TextView</code> 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 <code>TextView</code> storage. For example, a spell checker might use this method to replace a misspelled word with the correct spelling.
<code>OnShouldEndEditing</code>	Event called when editing should end.

## 2.9.2 Properties

---

### Overview

#### PUBLISHED PROPERTIES

Property name	Description
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.
<a href="#">TextInputTraits</a>	The <code>TextInputTraits</code> property defines features that are associated with keyboard input.
Visible	Shows / hides the TextView.

#### PUBLIC PROPERTIES

Property name	Description
TextView	Returns a reference to the native iOS <code>UITextView</code> .

---

## TextInputTraits

Property name	Description
AutoCapitalizationType	This property determines at what times the <code>⌘ Shift</code> key is automatically pressed, thereby making the typed character a capital letter. The default value for this property is <code>actTextAutocapitalizationTypeSentences</code> .
AutoCorrectionType	This property determines whether auto-correction is enabled or disabled during typing.
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.
EnablesReturnKeyAutomatically	A <code>Boolean</code> value indicating whether the return key is automatically enabled when text is entered by the users
KeyboardAppearance	The appearance style of the keyboard that is associated with the <code>TextView</code> .
KeyboardType	The keyboard style associated with the <code>TextView</code> .
ReturnKeyType	The contents of the <code>"return"</code> key.
SecureTextEntry	Identifies whether the <code>TextView</code> should hide the text being entered.

[Go back to Properties](#)

## 2.10 TMSFMXNativeUILabel

---

### 2.10.1 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.

### 2.10.2 Published Properties

Property name	Description
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 <code>1</code> . For Linbreak to work, <code>NumberOfLines</code> need to be larger than <code>1</code> .
TextColor	The color of the text.
Visible	Shows / hides the Label.

### 2.10.3 Public Properties

Property name	Description
Lbl	Returns a reference to the native iOS <code>UILabel</code> .

---

## 2.11 TMSFMXNativeUIScrollView

---

### 2.11.1 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.

### 2.11.2 Published Properties

Property name	Description
<code>AlwaysBounceHorizontal</code>	A Boolean value that determines whether bouncing always occurs when horizontal scrolling reaches the end of the content view or not.
<code>AlwaysBounceVertical</code>	A Boolean value that determines whether bouncing always occurs when vertical scrolling reaches the end of the content or not.
<code>Bounces</code>	Bounces the view horizontally or vertically depending on the <code>AlwaysBounceHorizontal</code> and <code>AlwaysBounceVertical</code> properties.
<code>DirectionalLockEnabled</code>	A Boolean value that determines whether scrolling is disabled in a particular direction or not.
<code>Enabled</code>	A Boolean value that determines whether scrolling is enabled or not.
<code>ShowsHorizontalScrollIndicator</code>	A Boolean value that controls whether the horizontal scroll indicator is visible or not.
<code>ShowsVerticalScrollIndicator</code>	A Boolean value that controls whether the vertical scroll indicator is visible or not.
<code>ViewForZooming</code>	View of the UIScrollView used for zooming.
<code>Visible</code>	Shows / hides the UIScrollView.

### 2.11.3 Public Properties

Property name	Description
<code>ScrollView</code>	Returns a reference to the native iOS <code>UIScrollView</code> .

### 2.11.4 Events

Event name	Description
<code>OnViewForZoomingInScrollView</code>	Returns a reference to a TMS FMX Native iOS Control used for zooming.



## 2.12 TMSFMXNativeUIProgressView

---

### 2.12.1 Usage

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

### 2.12.2 Published Properties

Property name	Description
Progress	The current progress of the ProgressView. The progress is a single value between <code>0.0</code> and <code>1.0</code> .
Style	The graphical style of the ProgressView.
Visible	Shows / hides the ProgressView.

### 2.12.3 Public Properties

Property name	Description
ProgressView	Returns a reference to the native iOS <code>UIProgressView</code> .

### 2.12.4 Events

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

## 2.13 TMSFMXNativeUISegmentedControl

---

### 2.13.1 Overview

---

#### 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.

#### Methods

Method name	Description
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

Event name	Description
OnValueChanged	Event called when the selected segment index has changed.

---

## 2.13.2 Properties

---

### Overview

#### PUBLISHED PROPERTIES

Property name	Description
<a href="#">Items (Segments)</a>	Collection of items (segments) in the SegmentedControl.
Style	The style of the SegmentedControl.
SelectedSegmentIndex	Sets or gets the <code>SelectedSegmentIndex</code> which is the index of an item in the <code>Items</code> collection.
TintColor	The tint color of the SegmentedControl.
Visible	Shows / hides the SegmentedControl.

#### PUBLIC PROPERTIES

Property name	Description
Button	Returns a reference to the native iOS <code>UISegmentedControl</code> .

**Items**

<b>Property name</b>	<b>Description</b>
Bitmap	The bitmap of an item (segment).
Enabled	Sets an item (segment) enabled or not.
Text	The text of an item (segment) in case there is no bitmap assigned.

[Go back to Properties](#)

## 2.14 TMSFMXNativeUIStepper

---

### 2.14.1 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.

### 2.14.2 Published Properties

Property name	Description
AutoRepeat	If <code>true</code> , the user pressing and holding on the stepper repeatedly alters value.
Continuous	If <code>true</code> , value change events are sent immediately when the value changes during user interaction. If <code>false</code> , 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 <code>true</code> , incrementing beyond <code>maximumValue</code> sets value to <code>minimumValue</code> ; likewise, decrementing below <code>minimumValue</code> sets value to <code>maximumValue</code> . If <code>false</code> , the Stepper does not increment beyond <code>maximumValue</code> nor does it decrement below <code>minimumValue</code> but rather holds at those values.

### 2.14.3 Public Properties

Property name	Description
Stepper	Returns a reference to the native iOS <code>UIStepper</code> .

### 2.14.4 Events

Property name	Description
OnValueChanged	Event called when the value of the stepper has changed.

## 2.15 TMSFMXNativeUITextField

---

### 2.15.1 Overview

#### 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.

#### Events

Event name	Description
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 <code>TextView</code> 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 <code>TextView</code> 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.

## 2.15.2 Properties

---

### Overview

#### PUBLISHED PROPERTIES


Property name	Description
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.
<a href="#">TextInputTraits</a>	The TextInputTraits property defines features that are associated with keyboard input.
Visible	Shows / hides the TextField.

#### PUBLIC PROPERTIES

Property name	Description
TextField	Returns a reference to the native iOS <code>UITextField</code> .

---

## TextInputTraits

Property name	Description
AutoCapitalizationType	This property determines at what times the  key is automatically pressed, thereby making the typed character a capital letter. The default value for this property is <code>actTextAutocapitalizationTypeSentences</code> .
AutoCorrectionType	This property determines whether auto-correction is enabled or disabled during typing.
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.
EnablesReturnKeyAutomatically	A Boolean value indicating whether the return key is automatically enabled when text is entered by the user.
KeyboardAppearance	The appearance style of the keyboard that is associated with the TextField.
KeyboardType	The keyboard style associated with the TextField.
ReturnKeyType	The contents of the <code>"return"</code> key.
SecureTextEntry	Identifies whether the TextField should hide the text being entered.

[Go back to Properties](#)



## 2.16 TMSFMXNativeMKMapView

---

### 2.16.1 Overview

---

#### 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.

## Methods

Method name	Description
AddAnnotation(ALatitude, ALongitude: Double): TTMSFMXNativeMKAnnotation	Adds and returns a new annotation based on a <code>Latitude</code> and <code>Longitude</code> .
AddAnnotation(ALatitude, ALongitude: Double; ATitle, ASubTitle: String): TTMSFMXNativeMKAnnotation	Adds and returns a new annotation based on a <code>Latitude</code> , <code>Longitude</code> , <code>Title</code> and <code>SubTitle</code> .
AddAnnotation(ALocation: TTMSFMXNativeMKMapLocation): TTMSFMXNativeMKAnnotation	Adds and returns a new annotaton based on a location.
AddAnnotation(ALocation: TTMSFMXNativeMKMapLocation; ATitle, ASubTitle: String): TTMSFMXNativeMKAnnotation	Adds and returns a new annotation based on a <code>location</code> , <code>Title</code> and <code>SubTitle</code> .
AddCircle(ALatitude, ALongitude: Double; ARadius: Double): TTMSFMXNativeMKOverlay	Adds and returns a new circle overlay shape based on a <code>Latitude</code> , <code>Longitude</code> and <code>Radius</code> parameter. The <code>Radius</code> parameter is in meters.
AddCircle(ALocation: TTMSFMXNativeMKMapLocation; ARadius: Double): TTMSFMXNativeMKOverlay	Adds and returns a new circle overlay shape based on a <code>Latitude</code> , <code>Longitude</code> and <code>Radius</code> parameter. The <code>Radius</code> parameter is in meters.
AddPolygon(ALocations: array of TTMSFMXNativeMKMapLocation): TTMSFMXNativeMKOverlay	Adds and returns a new polygon overlay shape based on an array of <code>Locations</code> .
AddPolyline(ALocations: array of TTMSFMXNativeMKMapLocation):TTMSFMXNativeMKOverlay	Adds and returns a new polyline overlay shape based on an array of <code>Locations</code> .
DeSelectAnnotation(AAnnotation: TTMSFMXNativeMKAnnotation)	Deselects the annotation with or without animation.
function AddImage(AURL: String; ATopLeftLocation, ABottomRightLocation: TTMSFMXNativeMKMapLocation): TTMSFMXNativeMKOverlay;	Adds and returns a new overlay image at a specific topleft and bottomright coordinate. <b>(iOS 7 or later)</b>
function AddTiling(AURL: String): TTMSFMXNativeMKOverlay;	Adds and returns a new overlay object that supports rendering tiles from a specific tile server. <b>(iOS 7 or later)</b>
GetAnnotation(Annotation: MKAnnotation): TTMSFMXNativeMKAnnotation	Returns the annotation collection item based on the native <code>MKAnnotation</code> .
GetOverlay(Overlay: MKOverlay): TTMSFMXNativeMKOverlay	Returns the overlay collection item based on the native <code>MKOverlay</code> .
GetRegion: TTMSFMXNativeMKMapRegion;	Returns the current region.
GetUserLocation: TTMSFMXNativeMKMapLocation	Returns the current user location

Method name	Description
procedure GetDirections(AStartLocation, AEndLocation: TTMSFMXNativeMKMapLocation; AAlternateRoutes: Boolean = False; ATransportType: TTMSFMXNativeMKMapViewDirectionsTransportType = ttDirectionsTransportTypeAutomobile; ADepartureDate: TDateTime = -1; AArrivalDate: TDateTime = -1);	Starts a get directions request with a given start and end location and optional parameters such as the possibility to calculate alternate routes and a departure and arrival date. When calling this method, the OnGetDirections event is triggered. When an error occurred during the request, the <a href="#">OnGetDirectionsError</a> event is triggered. <p>There is a second overload that accepts latitude and longitude parameters for both the start and the end location as doubles. (iOS 7 or later)</p>
RemoveAllAnnotations	Removes all annotations from the collection and the MapView.
RemoveAllOverlays	Removes all the overlays from the collection and the MapView
RemoveAnnotation(AAnnotation: TTMSFMXNativeMKAnnotation)	Removes a specific annotation from the collection and the MapView.
RemoveOverlay(AOverlay:TTMSFMXNativeMKOverlay)	Removes a specific overlay from the collection and the MapView.
SelectAnnotation(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: Boolean)	Sets the visible region of the MapView with TopLeft and BottomRight coordinates.
XYToCoordinate(X, Y: Single): TTMSFMXNativeMKMapLocation	Returns a latitude and longitude of an <code>X</code> and <code>Y</code> coordinate on the map based on the current region of the MapView.
ZoomToFitAnnotations(AIncludeUserLocation: Boolean = false; ALatitudeSpanOffset: Double = 0; ALongitudeSpanOffset: Double = 0);	Zoom the mapview to show/fit all annotations with optional parameters to include user location and additional region span offset.

## Events

Property name	Description
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.
OnGetDirections	Event called when a directions request is finished and successfully found one or multiple routes.
OnGetDirectionsError	Event called when a directions request is finished and an error occurred during the request.
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: TMSFMXNativeMKMapLocation;
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:

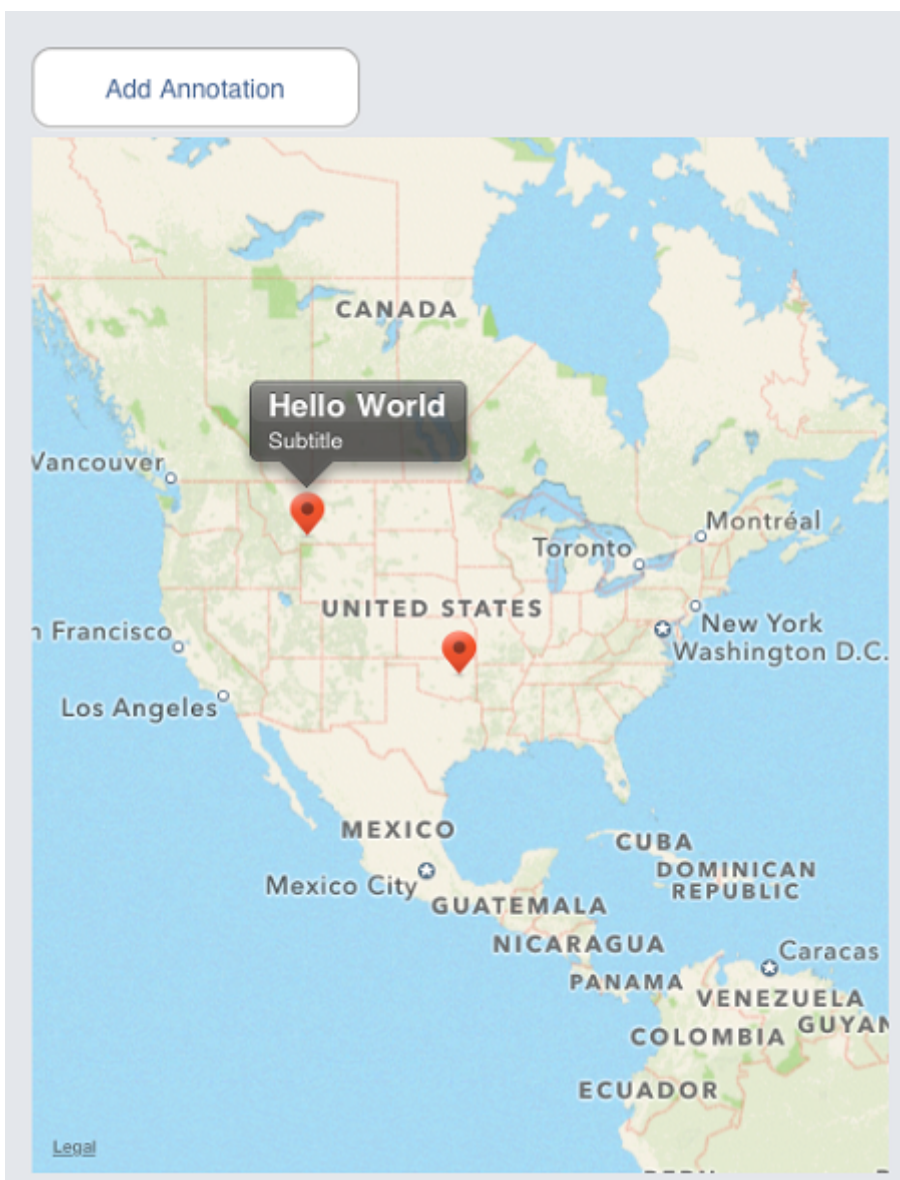


If you pan the Map, and click the button, the pin will be dropped in the center of the MapView again, but on a different coordinate. The `XYToCoordinate` functionality is using the current region, and zooming level of the MapView to return the correct coordinate.

## 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: TMSFMXNativeMKMapViewLocation;
  ann: TMSFMXNativeMKAnnotation;
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');
```



## Adding Overlays

Overlays are shapes that can be added to the map, there are different kinds of overlays, and each overlay can be configured in stroke and fill color and opacity. The `MapView` exposes an overlay collection and a couple of functions that can be used to add an overlay to the map. Currently, a circle, polygon and polyline can be added to the `MapView`. Below is a sample of a circle and a polyline that is added to the `MapView`. Note that the code is wrapped with a `BeginUpdate` and `EndUpdate`. This is crucial to make sure the overlay shape has the correct position, and appearance when added.

```
//Center circle with 500 km radius
TMSFMXNativeMKMapView1.BeginUpdate;
c :=
TMSFMXNativeMKMapView1.AddCircle(TMSFMXNativeMKMapView1.XYToCoordinate(TMSFMXNativeMKMapView1.Width /
2,
    TMSFMXNativeMKMapView1.Height / 2), 500000);
c.LineWidth := 3;
c.LineColor := TAlphaColorRec.Greenyellow;
c.Color := TAlphaColorRec.Darkgoldenrod;
c.Opacity := 0.5;
c.LineOpacity := 0.5;
TMSFMXNativeMKMapView1.EndUpdate;

//Bermuda triangle
TMSFMXNativeMKMapView1.BeginUpdate;
arr[0].Latitude := 25.774252;
arr[0].Longitude := -80.190262;
arr[1].Latitude := 18.466465;
arr[1].Longitude := -66.118292;
arr[2].Latitude := 32.321384;
arr[2].Longitude := -64.75737;
arr[3] := arr[0];

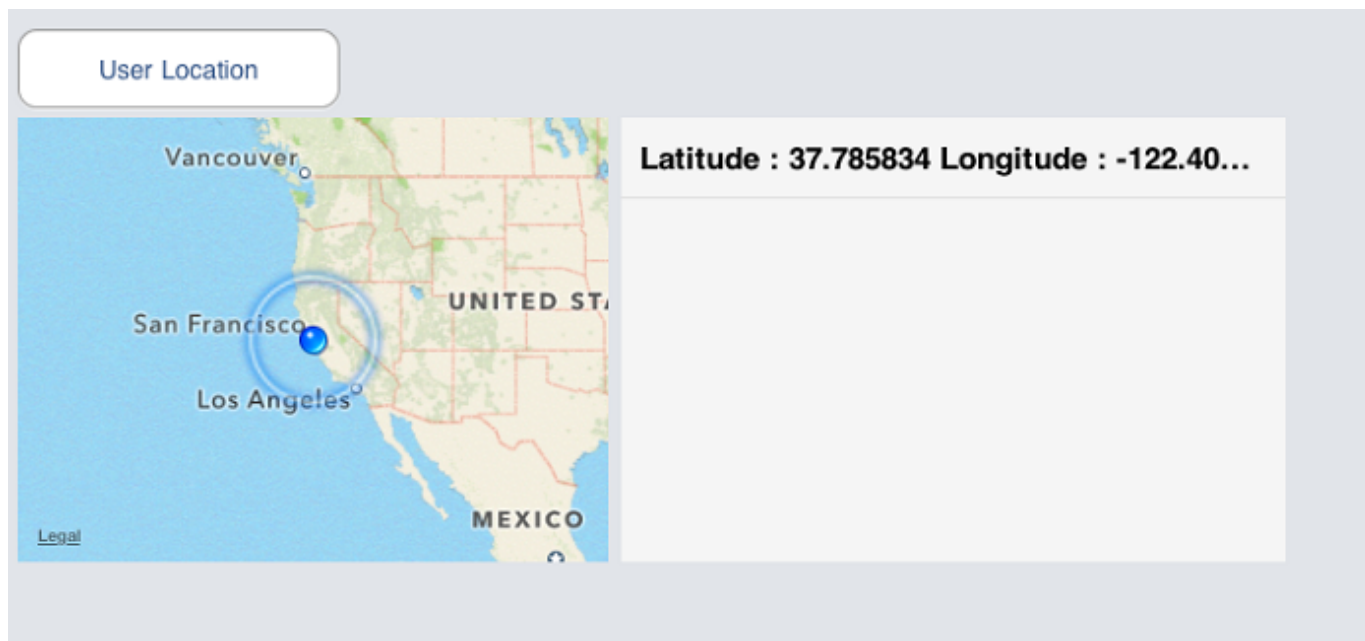
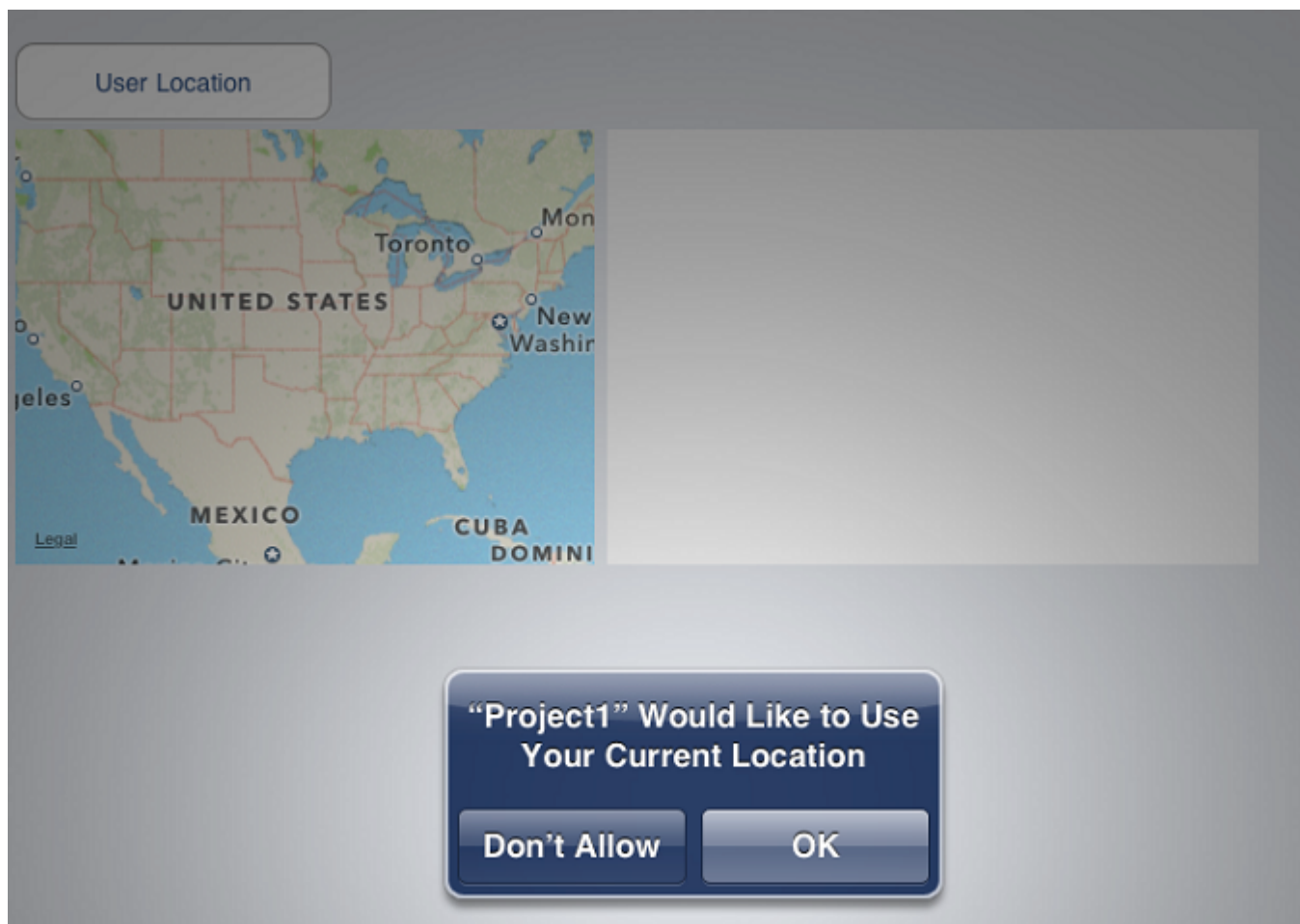
c := TMSFMXNativeMKMapView1.AddPolyline(arr);
c.LineColor := TAlphaColorRec.Red;
TMSFMXNativeMKMapView1.EndUpdate;
```

## 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.

## Directions (iOS 7 or later)

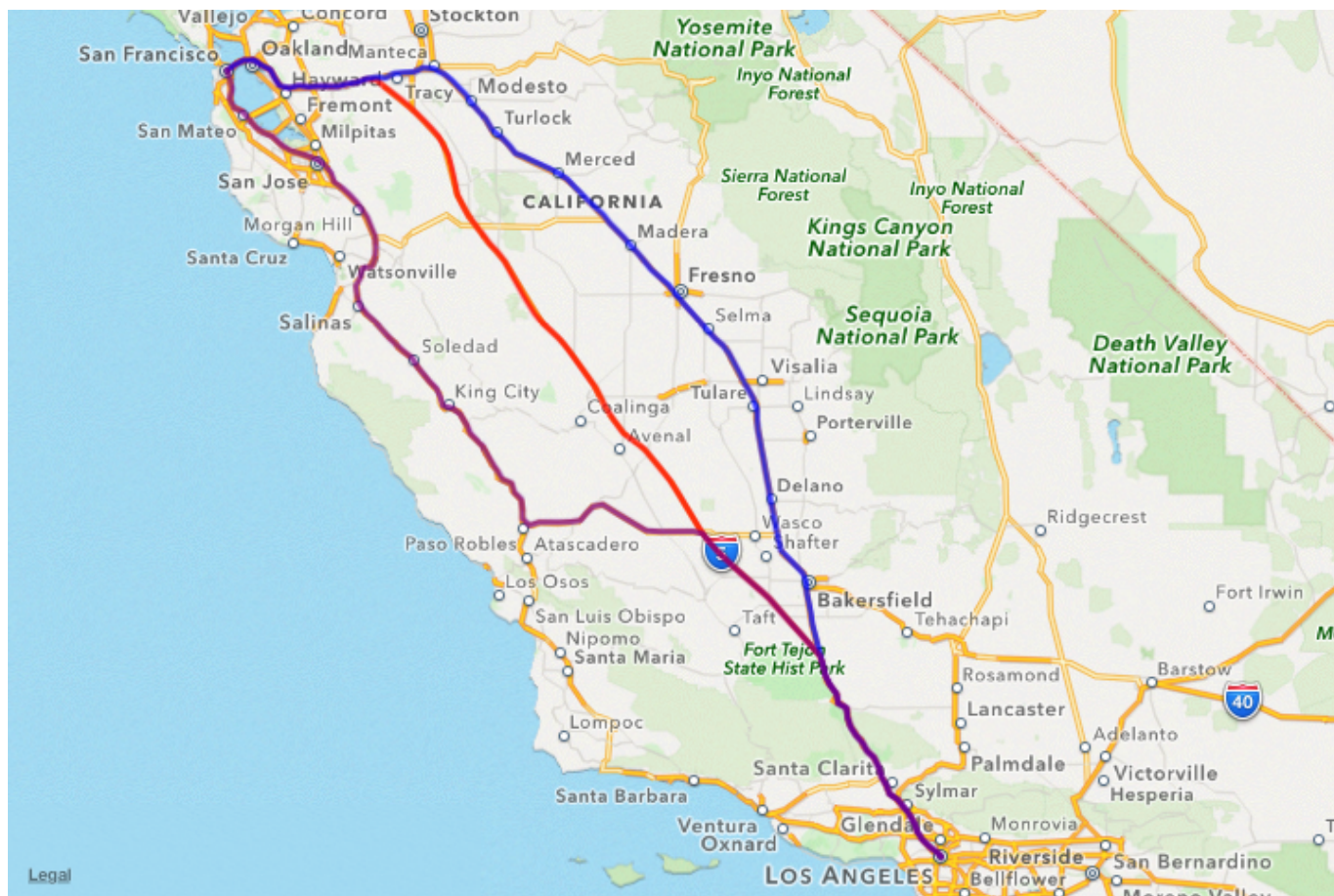
The MapView has built-in functionality to calculate directions between 2 locations. Below is a sample between San Francisco and Los Angeles with the optional parameter to calculate alternative routes.

```
TMSFMXNativeMKMapView1.GetDirections(37.774929499999999, -122.4194155, 34.054434999999998, -118.253393, true);
```

After executing this request, the `OnGetDirections` event is called with a `TTMSFMXNativeMKDirectionsResponse` record which contains information about the routes that were found. To visualize this data, you can add a polyline to the map that accepts an array of locations with the following code:

```
procedure TForm1.TMSFMXNativeMKMapView1GetDirections(Sender: TObject;
  AResponse: TTMSFMXNativeMKDirectionsResponse);
var
  r: Integer;
  pl: TTMSFMXNativeMKOverlay;
begin
  TMSFMXNativeMKMapView1.BeginUpdate;
  for r := 0 to Length(AResponse.Routes) - 1 do
  begin
    pl := TMSFMXNativeMKMapView1.AddPolyline(AResponse.Routes[r].Locations);
    pl.LineOpacity := 0.75;
    if r = 0 then
      pl.LineColor := TAlphaColorRec.Red
    else if r = 1 then
      pl.LineColor := TAlphaColorRec.Blue
    else if r = 2 then
      pl.LineColor := TAlphaColorRec.Purple;
  end;
  TMSFMXNativeMKMapView1.EndUpdate;
end;
```

This gives the following result:



### Tiles (iOS 7 or later)

The `MapView` also supports the `MKTileOverlay`, which implements an overlay that is optimized for covering an area of the map using individual bitmap tiles. The bitmap tiles are provided by a server through a specific URL. Below is a sample before and after applying Google Maps and OpenStreetMap tiles with their specific formatted URL's.

```

procedure TForm1.TMSFMXNativeUIButton1Click(Sender: TObject);
begin
    TMSFMXNativeMKMapView1.BeginUpdate;
    TMSFMXNativeMKMapView1.AddTiling('http://mt1.google.com/vt/lyrs=m@110&hl=pl&x={x}&y={y}&z={z}');
    TMSFMXNativeMKMapView1.EndUpdate;
end;

procedure TForm1.TMSFMXNativeUIButton2Click(Sender: TObject);
begin
    TMSFMXNativeMKMapView2.BeginUpdate;
    TMSFMXNativeMKMapView2.AddTiling('http://tile.openstreetmap.org/{z}/{x}/{y}.png');
    TMSFMXNativeMKMapView2.EndUpdate;
end;

```

Google Maps



OpenStreetMap



Google Maps



OpenStreetMap



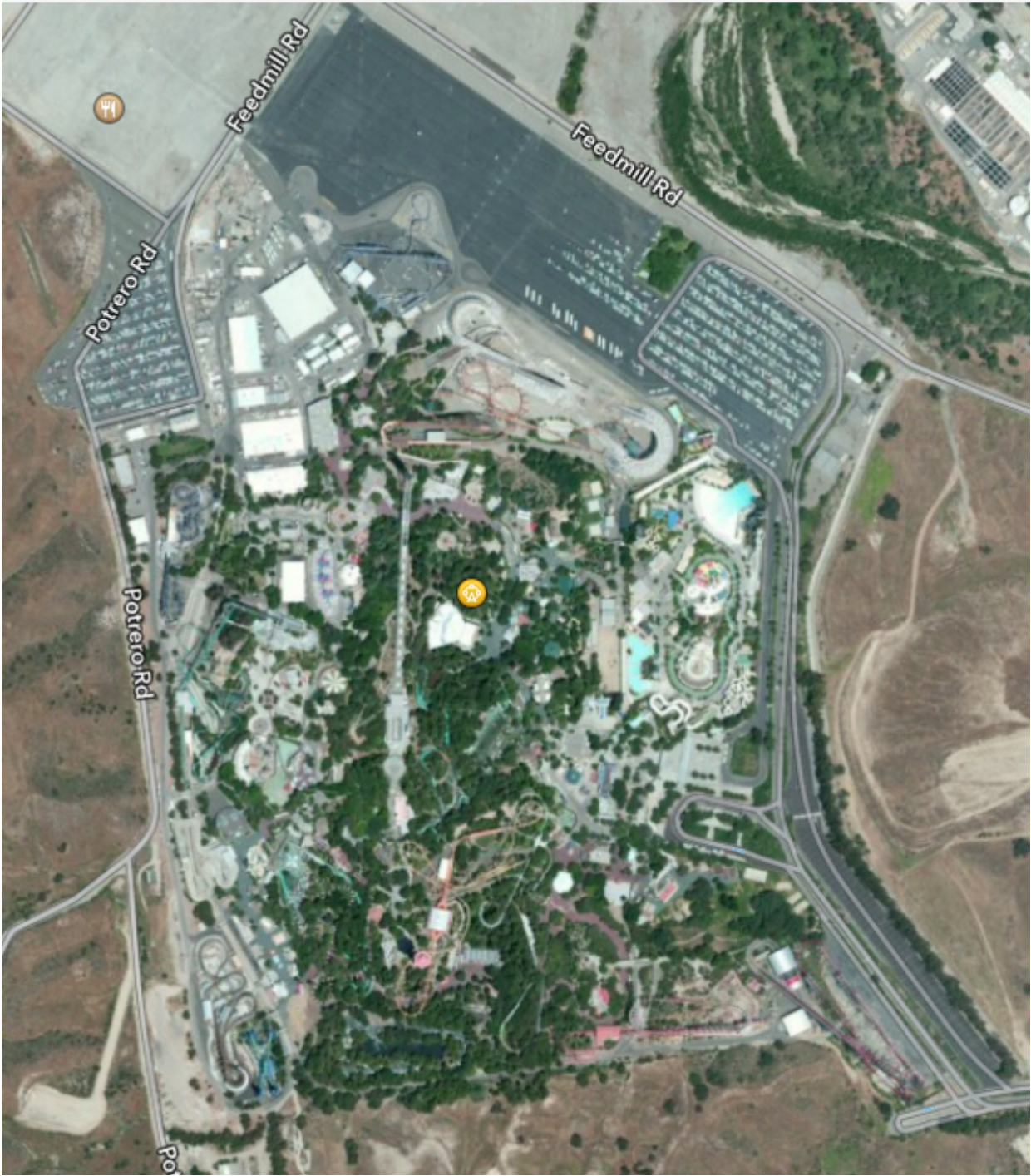
### Image overlay (iOS 7 or later)

When a specific area needs to be marked, you can add an image that is added on top of the map, bounded by a topleft and bottomright coordinate. Below is a sample that adds an image (added through the deployment window in your project) and renders it at the specific topleft and bottomright coordinate. It also centers and zooms the map on a specific coordinate to visualize the image.

```
TMSEMXNativeMKMapView1.BeginUpdate;
t1 := MakeMapLocation(34.4311, -118.6012);
```

```
tr := MakeMapLocation(34.4311, -118.5912);
bl := MakeMapLocation(34.4194, -118.6012);
br := MakeMapLocation(34.4194, -118.5912);
mr := MakeMapLocation(34.4248, -118.5971);
splat := Abs(br.Latitude - tl.Latitude);
splon := Abs(br.Longitude - tl.Longitude);
rgn.Center.Latitude := mr.Latitude;
rgn.Center.Longitude := mr.Longitude;
rgn.Span.latitudeDelta := splat;
rgn.Span.longitudeDelta := splon;
TMSFMXNativeMKMapView1.SetRegion(rgn, True);
TMSFMXNativeMKMapView1.AddImage(ExtractFilePath(ParamStr(0)) + 'overlay_park.png', tl, br);
TMSFMXNativeMKMapView1.MapType := mtMapTypeHybrid;
TMSFMXNativeMKMapView1.EndUpdate;
```

The result of the code is shown in the screen below, with and without the image to demonstrate the difference. Comment out the line `TMSFMXNativeMKMapView1.MapType := mtMapTypeHybrid;` to have a better view of the added image.



## 2.16.2 Properties

### Overview

#### PUBLISHED PROPERTIES

Property name	Description
<a href="#">Annotations[Index]</a>	A collection of annotations, used to annotate the map and display custom information through annotations pins.
MayType	The type of data displayed by the MapView.
<a href="#">Overlays[Index]</a>	A collection of overlay shapes (circles, polylines and polygons).
ScrollEnabled	Enables / disables scrolling on the MapView.
ShowsUserLocation	Shows / hides the current user location.
UserTrackingMode	The user tracking mode that centers the map on the user location. Optional heading direction that automatically rotates the map in the user heading direction.
Visible	Shows / hides the MapView.
ZoomEnabled	Enables / disables zooming on the MapView.

#### PUBLIC PROPERTIES

Property name	Description
MapView	Returns a reference to the native iOS <code>MKMapView</code> .

## Annotations

Property name	Description
AnimatesDrop	Animates a drop of a pin when added to the MapView.
Bitmap	Shows an image for the view of an annotation. If a bitmap is assigned, the default pin is replaced with this image.
CanShowCallout	Enables or disables showing the default callout for an annotation, which shows a title, subtitle and / or callout accessory views.
Draggable	Enables or disables dragging of an annotation. When dragging an annotation the <code>OnMapViewAnnotationDragStateChanged</code> event is called with various drag states.
Enabled	Enables or disables interaction with an annotation.
LeftCalloutAccessoryView	The left callout accessory view of an annotation. This view can be linked to another instance of a TMS FMX Native UI Control.
PinColor	The color of the pin of an annotation.
RightCalloutAccessoryView	The right callout accessory view of an annotation. This view can be linked to another instance of a TMS FMX Native UI Control.
SubTitle	The sub title of an annotation shown in the callout.
Title	The title of an annotation shown in the callout. If the title is empty, no callout is shown.

[Go back to Properties](#)



## Overlays

Property name	Description
CanReplaceMapContent	If <code>true</code> , replaces the current drawn MapView tiles with your custom tiles. If <code>false</code> , draws the custom tiles on top of the default MapView tiles. Opacity can be specified for an overlay object, or can be included in the tile images that are rendered after adding them through the tile overlay template URL. (iOS 7 or later)
Color	The color of the overlay shape.
GeometryFlipped	Indicates the orientation of tile indexes along the y axis. (iOS 7 or later)
Kind	The kind of shape (circle, polyline, polygon)
Level	The level that is used when adding the tiles. <ul style="list-style-type: none"> <li>- <code>tlAboveRoads</code>: Place the overlay above roadways but below map labels, shields, or point-of-interest icons.</li> <li>- <code>tlAboveLabels</code>: Place the overlay above map labels, shields, or point-of-interest icons but below annotations and 3D projections of buildings. (iOS 7 or later)</li> </ul>
LineColor	The color of the border / line of the overlay shape.
LineOpacity	The opacity of the border / line of the overlay shape. Value from <code>0</code> to <code>1</code> .
LineWidth	The width of the border / line of the overlay shape.
Locations	A collection of locations with a latitude and longitude. Used in combination with polyline, polygon overlay shapes.
MaximumZoomLevel	The maximum zoom level supported by the tiles rendered by the overlay object. (iOS 7 or later)
MinimumZoomLevel	The minimum zoom level supported by the tiles rendered by the overlay object. (iOS 7 or later)
Opacity	The opacity of the overlay shape. Value from <code>0</code> to <code>1</code> .
Radius	The radius in meters of a circle overlay shape.
TileSize	The tile size used when rendering the custom tiles. The default size is 256 x 256. (iOS 7 or later)
URL	The url template or file used for adding a tile overlay or image on the MapView. (iOS 7 or later)

[Go back to Properties](#)

## 2.17 TMSFMXNativeCLGeoCoder

---

### 2.17.1 Usage

The `TMSFMXNativeCLGeoCoder` component provides services for converting between a coordinate (specified as a latitude and longitude) and the user-friendly representation of that coordinate. A user-friendly representation of the coordinate typically consists of the street, city, state, and country information corresponding to the given location, but it may also contain a relevant point of interest, landmarks, or other identifying information.

### 2.17.2 Methods

Method name	Description
<code>GetGeocodeLocation</code>	Overload that accepts a location record or latitude and longitude double and starts a request to translate the location to an array of placemarks which contain information about the requested location.
<code>GetReverseGeocodeLocation</code>	Accepts an address string and starts a request to translate to address to an array of placemarks which contain information about the requested address.

### 2.17.3 Events

Event name	Description
<code>OnGetGeocodeLocation</code>	Event called when an address is passed to the <code>GetGeocodeLocation</code> method and placemarks for that address are found. A placemark holds the location for that address. Multiple placemarks can be found for one address
<code>OnGetGeocodeError</code>	Event called when an Address is passed to the <code>GetGeocodeLocation</code> and an error occurred during the request.
<code>OnGetReverseGeocodeLocation</code>	Event called when a location is passed to the <code>GetReverseGeocodeLocation</code> and placemarks for that location are found. A placemark holds the address for that location. Multiple placemarks can be found for one location.
<code>OnGetReverseGeocodeError</code>	Event called when a location is passed to the <code>GetReverseGeocodeLocation</code> method and an error occurred during the request.

## 2.18 TMSFMXNativeFMXWrapper

---

### 2.18.1 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.

### 2.18.2 Published Properties

---

Property name	Description
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.

---

## 2.19 TMSFMXNativeUIImageView

### 2.19.1 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`

### 2.19.2 Properties

Property name	Description
Bitmap	Sets the image of an UIImageView.
BitmapFile	A direct link to an image file located in the root or documents directory.
BitmapLink	A link to another <code>TBitmap</code> 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 UIImageView.
URL	Loads an Image from an URL.

### 2.19.3 Methods

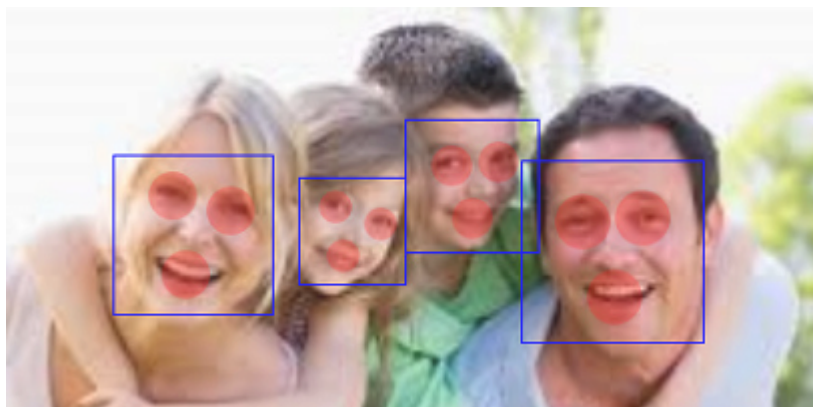
Method name	Description
DetectFaces	Method which detect faces in an image. Each face is stored in the Faces collection.
ShowFaces	Method which detects and marks faces on the image. Each face is stored in the Faces collection

## 2.19.4 Public Properties

Property name	Description
DefaultSizeFactor	The factor that is applied to the size of the face in order to create a left eye, right eye and mouth rectangle part. These values can be retrieved through the Faces collection.
Faces	A collection of faces when calling <code>DetectFaces</code> or <code>ShowFaces</code> . Each face contains information about the position of the left eye, right eye and mouth part. Starting from iOS7, the information can also contain if the eye is closed, the face is smiling and the face angle
ImageView	Returns a reference to the native iOS UIImageView.
LeftEyeColor	The color to indicate the left eye after detection of the face.
MouthColor	The color to indicate the mouth after detection of the face.
RightEyeColor	The color to indicate the right eye after detection of the face.

## 2.19.5 Face Detection

The ImageView supports face detection when an appropriate image is loaded. Call `DetectFaces` to fill the Faces collection or call `ShowFaces` to fill the collection and display a rectangle for each face in combination with the left eye, right eye and mouth part ellipses. Additionally parameters can be passed to the `DetectFaces` or `ShowFaces` call to allow a lower accuracy, a minimum face size to detect and / or an image orientation from which to start searching. Starting from iOS 7, parameters can be added to detect an eye blink and / or a smile. Below is a sample that demonstrates calling the default `ShowFaces` call on an image:



## 2.20 TMSFMXNativeUIPopoverController

---

### 2.20.1 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.

### 2.20.2 Published Properties

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

### 2.20.3 Public Properties

Property name	Description
PopOver	Returns a reference to the native iOS <code>UIPopoverController</code> .

### 2.20.4 Methods

Property name	Description
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)

## 2.21 TMSFMXNativeUIView

---

### 2.21.1 Usage

---

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

### 2.21.2 Published Properties

---

Property name	Description
Visible	Shows / hides the View.

---

### 2.21.3 Public Properties

---

Property name	Description
View	Returns a reference to the native iOS <code>UIView</code> .

---

### 2.21.4 Published Events

---

Property name	Description
OnDrawRect	Event to perform custom drawing inside the View.

---

## 2.22 TMSFMXNativeUIImagePickerController

---

### 2.22.1 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.

### 2.22.2 Published Properties

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

### 2.22.3 Public Properties

Property name	Description
<code>ImagePicker</code>	Returns a reference to the native iOS <code>UIImagePickerController</code> .
<code>Popover</code>	Returns a reference to the native iOS <code>UIPopoverController</code> .



## 2.22.4 Methods

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

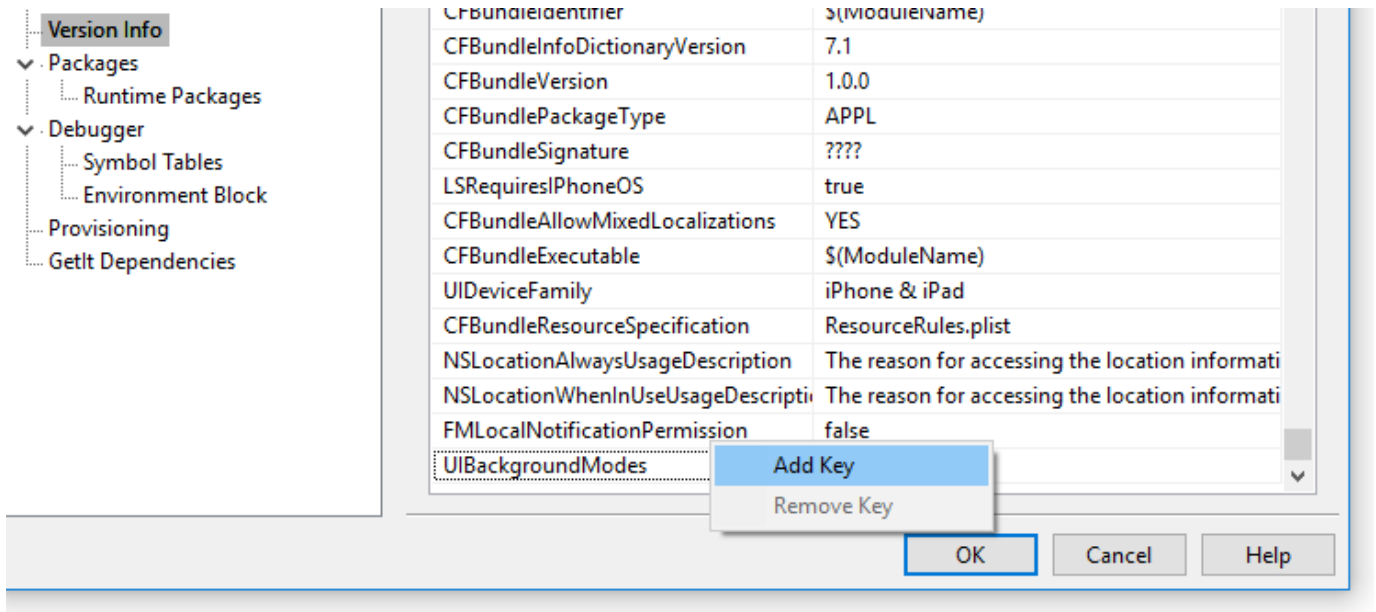
## 2.22.5 Public Events

Events name	Description
OnDidFinishPickingMediaWithInfo	Access directly to the media dictionary after taking an image or capturing a video with a native reference to an <code>NSDictionary</code> reference.
OnDidFinishPickingImage	Event called when taking an image with native access to a <code>UIImage</code> reference.

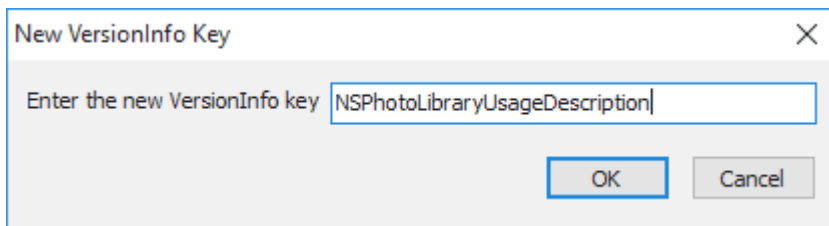
## 2.22.6 Events

Events name	Description
OnDidCancel	Event called when cancel is pressed.
OnDidFinishPickingBitmap	Event called when an image is taken or picked and the image is converted to a <code>TBitmap</code> .

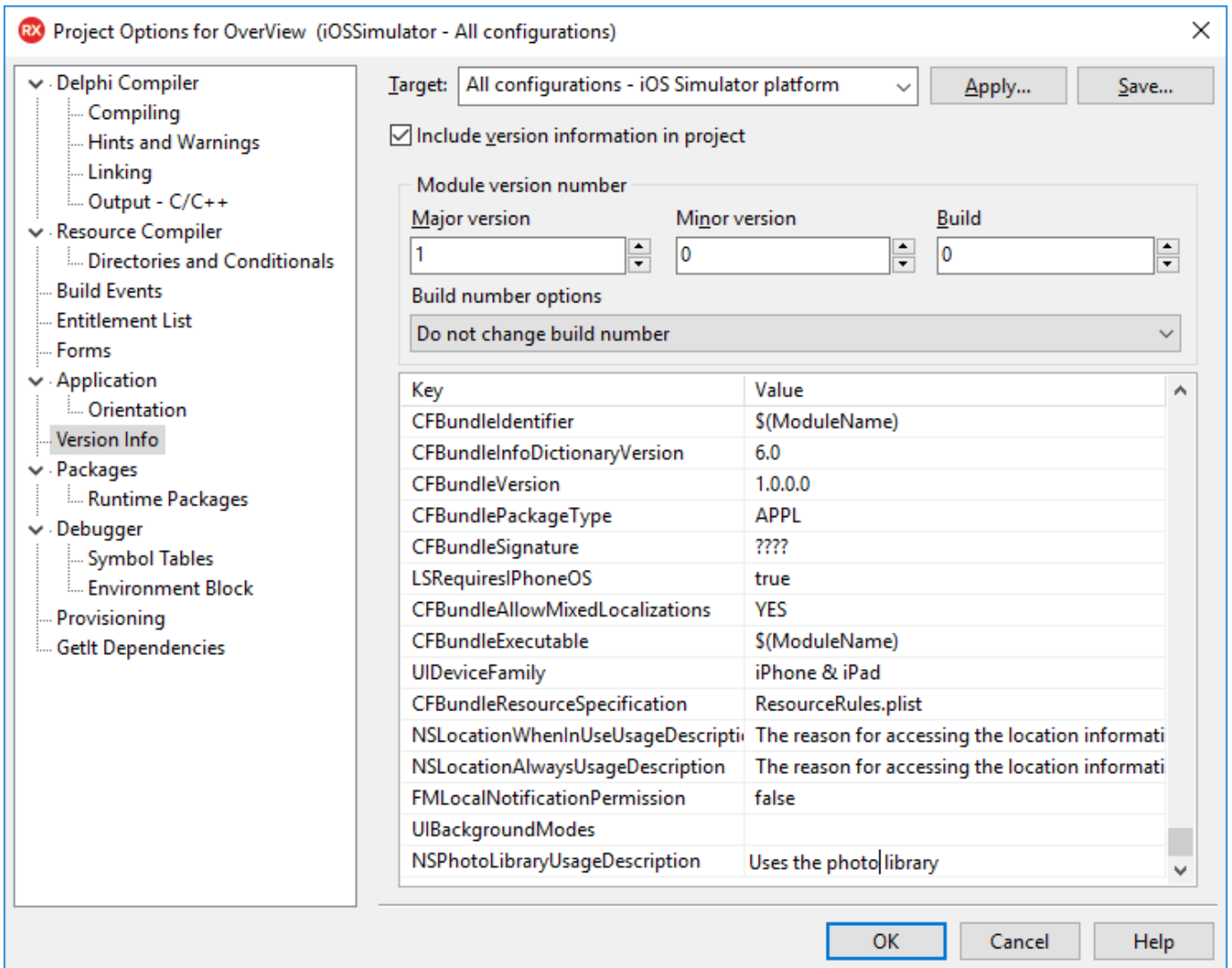
Starting from iOS 10 a new `NSPhotoLibraryUsageDescription` key is necessary on order to correctly initialize the photo library and prevent the application from crashing. This key needs to be added to each individual project. Start by going to the project options and go to version info. Scroll to the bottom, right-click and select "Add Key".



A dialog will popup, prompting for the new version info key. Fill in "NSPhotoLibraryUsageDescription".



After clicking "OK", the new entry still needs a value, which can be anything descriptive for your application. In the demo, we have added "Uses the photo library"



## 2.23 TMSFMXNativeUITabBarController

---

### 2.23.1 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.

### 2.23.2 Published Properties

Property name	Description
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.

### 2.23.3 Public Properties

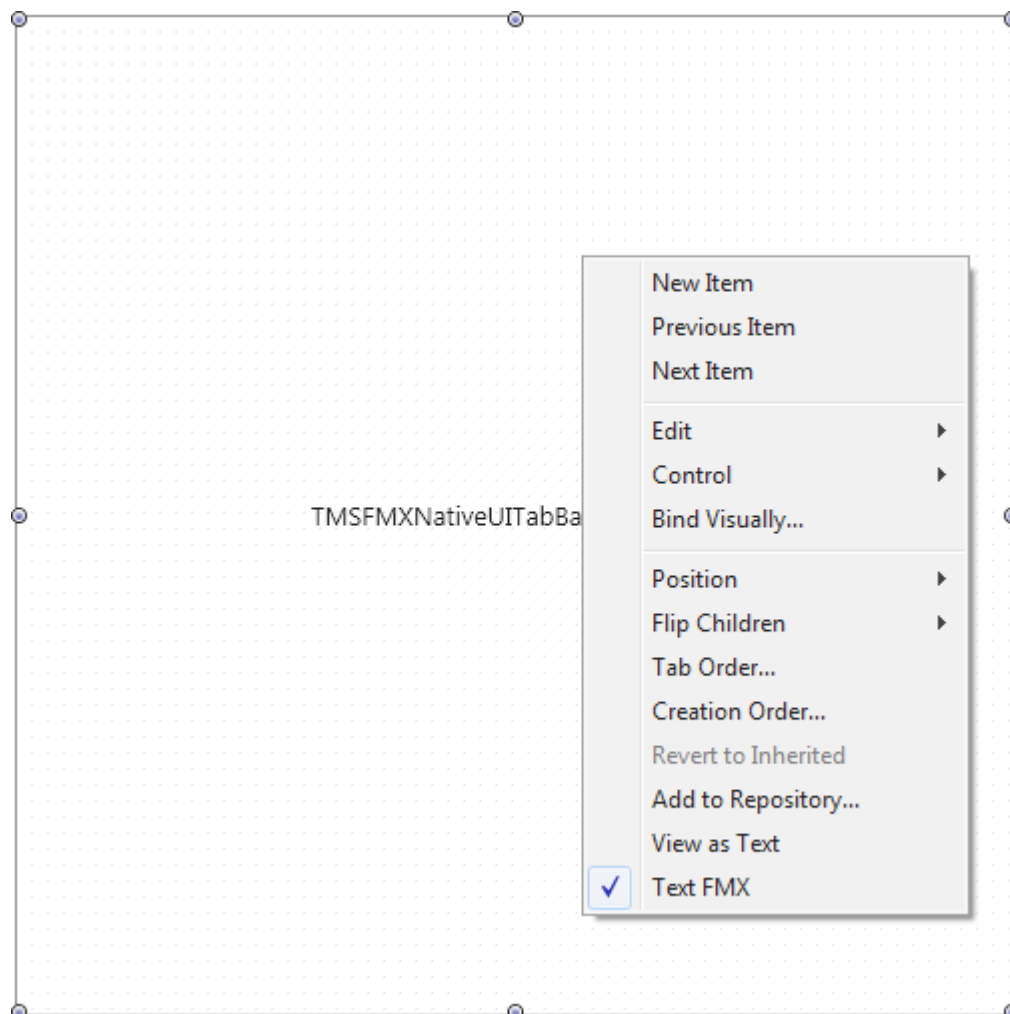
Property name	Description
TabBarController	Returns a reference to the native iOS <code>UITabBarController</code> .
ViewController (TabBarItem level)	Returns a reference to the native iOS <code>UIViewController</code> that is used for each tab inside the TabBarController.

### 2.23.4 Events

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

## 2.23.5 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.



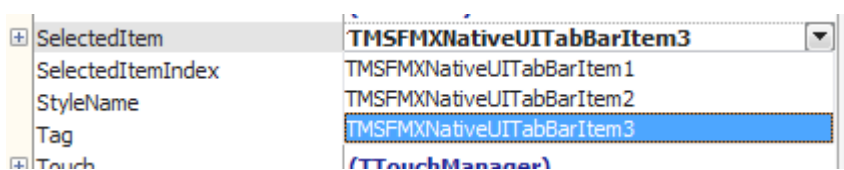
Clicking on `New Item`, adds a new `TMSFMXNativeUITabBarItem` instance to the form (TabBarItem), which can only be used inside the TabBar. In the sample below, 2 TabBarItems have been added, and the first TabBarItem is selected.

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



## 2.23.6 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.

## 2.24 TMSFMXNativeUINavigationController

---

### 2.24.1 Usage

The `UINavigationController` class implements a specialized view controller that manages the navigation of hierarchical content. This navigation interface makes it possible to present your data efficiently and also makes it easier for the user to navigate that content.

### 2.24.2 Published Properties

Property name	Description
Color	The background color of the UINavigationController.
Title	The title of the current page in the UINavigationController.
Visible	Shows / hides the UINavigationController.

### 2.24.3 Methods

Method name	Description
<code>pushViewController(AViewController:TTMSFMXNativeUIViewController; AAnimated: Boolean);</code>	Pushes a ViewController on the stack with optional animation
<code>popViewController</code>	Pops the last added ViewController on the stack with optional animation.

### 2.24.4 Public Properties

Property name	Description
<code>NavigationController</code>	Returns a reference to the native iOS <code>UINavigationController</code> .

### 2.24.5 Published Events

Property name	Description
<code>OnDrawRect</code>	Event to perform custom drawing inside the UINavigationController.

### 2.24.6 Pushing and popping pages (ViewControllers)

The UINavigationController can be filled with Pages and has a toolbar at the top. The pages can be pushed and popped from the main UINavigationController through code. You can add as many pages as you wish by adding a

`TTMSFMXNativeUIViewController` for each page. Below is a sample with 3 pages.

Drop an instance of `TMSFMXNativeUINavigationController` and 2 instances of `TMSFMXNativeUIViewController` on the form. Set the title for each controller like the sample below:

```
TMSFMXNativeUINavigationController1.Title := 'First Page';
TMSFMXNativeUIViewController1.Title := 'Second Page';
TMSFMXNativeUIViewController2.Title := 'Third Page';
```

Add a `TMSFMXNativeUIButton` instance as a child of the first page, the `TMSFMXNativeUINavigationController`. In the `OnClick` event, add the following code:

```
TMSFMXNativeUINavigationController1.PushViewController(TMSFMXNativeUIViewController1, True);
```

This will push the second page in place and update the toolbar with a back button. The back button will allow you to navigate to the previous page. This can also be done with the second page by dropping a button on the first page and adding the code:

```
TMSFMXNativeUINavigationController1.PushViewController(TMSFMXNativeUIViewController2, True);
```

If you press on the first button, and on the second button, the `NavigationController` will push 2 `ViewControllers` in place and update back button to return to the second page. The hierarchy is now updated with 3 pages. To return one step in the hierarchy, call

```
TMSFMXNativeUINavigationController1.PopViewController(True);
```



## 2.25 TMSFMXNativeUIViewController

---

### 2.25.1 Usage

---

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

### 2.25.2 Published Properties

---

Property name	Description
Visible	Shows / hides the ViewController.

---

### 2.25.3 Public Properties

---

Property name	Description
ViewController	Returns a reference to the native iOS <code>UIViewController</code> .

---

### 2.25.4 Published Events

---

Property name	Description
OnDrawRect	Event to perform custom drawing inside the ViewController.

---

## 2.26 TMSFMXNativeUIViewPopoverController

---

### 2.26.1 Usage

The TMSFMXNativeUIViewPopoverController class lets you present your view controller as a popover view.

### 2.26.2 Published Properties

Property name	Description
Color	The color of the view controller.
Title	The title of the view controller.

### 2.26.3 Public Methods

Method name	Description
GetParentView: UIView	Retrieves the parent view of the sheet controller.
ShowFromControl(AControl: TMSFMXNativeUIBaseControl)	Shows the popover view from a control.
ShowFromButton(AButton: UIBarButtonItem)	Shows the popover view from a UIBarButtonItem.
ShowFromRect(ARect: TRectF)	Shows the popover view from a rectangle on the parent view.
ShowFromRect(ARect: TRectF; AView: UIView)	Shows the popover view from a rectangle on the selected view.
Hide(AAnimated: Boolean)	Hide the sheet view.

## 2.27 TMSFMXNativeUISheetController

---

### 2.27.1 Usage

The TMSFMXNativeUISheetController class lets you present your view controller as a sheet.

### 2.27.2 Published Properties

Property name	Description
Detents	The possible sizes of the sheet.
SelectedDetent	The selected size the sheet should show.
ShowGrabber	Make the grabber on top of the sheet visible.
Color	The color of the view controller.
Title	The title of the view controller.

### 2.27.3 Public Methods

Method name	Description
GetParentView: UIView	Retrieves the parent view of the sheet controller.
ShowFromView(AView: UIView)	Shows the sheet from a view.
Show	Shows the sheet from the parent view.
Hide(AAnimated: Boolean)	Hide the sheet view.

## 2.28 TMSFMXNativeUIPageViewController

---

### 2.28.1 Usage

A page view controller lets the user navigate between pages of content, where each page is managed by its own view controller. Navigation can be controlled by the user using gestures. When navigating from page to page, the page view controller uses the transition that you specify to animate the change.

### 2.28.2 Published Properties

Property name	Description
NavigationDirection	The direction of navigation when using scrolling transition style.
NavigationOrientation	The orientation of navigation when using scrolling transition style.
PageControl	Displays a page control to indicate the number of pages and the current page index when using scrolling transition style.
Pages	A collection of pages connected to a View of type <a href="#">TMSFMXNativeUIViewController</a> .
PageSpacing	The spacing between pages.
SpineLocation	The location of the spine when using page curl transition style.
TransitionStyle	The style of the transition. The values are scrolling or page curl transition style.
Visible	Shows / hides the View.

### 2.28.3 Public Properties

Property name	Description
PageViewController	Returns a reference to the native iOS <a href="#">UIPageViewController</a> .

### 2.28.4 Public Events

Property name	Description
OnCustomizeViewForPage	Event called to customize the ViewController instance that is returned for a specific page.

## 2.28.5 Published Events

---

<b>Property name</b>	<b>Description</b>
OnGetNumberOfPages	Event to return the number of pages in a PageViewController.
OnGetViewForPage	Event to return the view for a specific page.

---

## 2.29 TMSFMXNativeUIPDFPageViewController

---

### 2.29.1 Usage

Inherits from `TMSFMXNativeUIViewController` and draws a single PDF page based on a `Location` and `PageIndex` property.

### 2.29.2 Published Properties

Property name	Description
Location	The location of the PDF file.
PageIndex	The index of the page that needs to be drawn. The <code>PageIndex</code> property starts from 1
Visible	Shows / hides the ViewController.

### 2.29.3 Public Properties

Property name	Description
ViewController	Returns a reference to the native iOS <code>UIViewController</code> .

### 2.29.4 Published Events

Property name	Description
OnDrawRect	Event to perform custom drawing inside the ViewController.

## 2.30 TMSFMXNativeUIPDFViewController

---

### 2.30.1 Usage

---

Inherits from `TMSFMXNativeUIPageViewController` and uses the `TMSFMXNativeUIPDFPageViewController` class to draw all pages of a single PDF File.

### 2.30.2 Published Properties

---

Property name	Description
Location	The location of the PDF file.
Visible	Shows / hides the PDFViewController.

---

### 2.30.3 Public Properties

---

Property name	Description
PageViewController	Returns a reference to the native iOS <code>UIPageViewController</code> .

---

## 2.31 TMSFMXNativeUIActionSheet

---

### 2.31.1 Usage

Use the `UIActionSheet` class to present the user with a set of alternatives for how to proceed with a given task.

### 2.31.2 Published Properties

Property name	Description
Buttons	A string list of additional buttons.
CancelButtonTitle	The title of the cancel button.
DestructiveButtonTitle	The title of the destructive button.
Style	The style of the ActionSheet.

### 2.31.3 Methods

Method name	Description
ShowFromTabBar	Shows the ActionSheet from a tabBar.
ShowFromControl	Shows the ActionSheet from a control.
ShowFromButton	Shows the ActionSheet from a toolbar button.
ShowFromRect	Shows the ActionSheet in a specific rectangle relative to the root view.
ShowFromRectInView	Shows the ActionSheet in a specific rectangle relative to the view passed as a parameter.
ShowFromToolBar	Shows the ActionSheet from a toolbar.

**Note: All ShowFrom\* methods have the same result on an iphone application. The ActionSheet is presented from the bottom of the screen.**



## 2.31.4 Public functions

Property name	Description
CancelButtonIndex: Integer;	Returns the index linked to the CancelButtonTitle.
DestructiveButtonIndex: Integer;	Returns the index linked to the DestructiveButtonTitle.
FirstOtherButtonIndex: Integer;	Returns the first index of the additional Button titles added to the Buttons string list property.
NumberOfButtons: Integer;	Returns the number of buttons displayed in an ActionSheet.
ButtonTitleAtIndex(AIndex: Integer);	Returns the title of a specific button.

## 2.31.5 Public Properties

Property name	Description
ActionSheet	Returns a reference to the native iOS <code>UIView</code> .

## 2.31.6 Published Events

Property name	Description
OnCancel	Event triggered when cancelling the ActionSheet.
OnClickedAtButtonIndex	Event triggered when a specific button index is clicked.
OnDidDismissWithButtonIndex	Event triggered when the ActionSheet is dismissed with a specific button index.
OnDidPresent	Event called when the ActionSheet is presented.
OnWillDismissWithButtonIndex	Event called when the ActionSheet will be dismissed with a specific button index.
OnWillPresent	Event called when the ActionSheet will be presented.

## 2.32 TMSFMXNativeMFMailComposeViewController

---

### 2.32.1 Usage

The `MFMailComposeViewController` class provides a standard interface that manages the editing and sending an email message. You can use this view controller to display a standard email view inside your application and populate the fields of that view with initial values, such as the subject, email recipients, body text, and attachments. The user can edit the initial contents you specify and choose to send the email or cancel the operation.

### 2.32.2 Published Properties

Property name	Description
Attachments	A string list of file locations that need to be attached to the mail.
BccRecipients	A string list of Bcc recipients.
Body	The body of the mail. Can either be passed as plain or HTML text. Can be used in combination with the <code>IsHTML</code> property.
CcRecipients	A string list of Cc recipients.
IsHTML	Enables or disables whether the body needs to be sent as plain or HTML text.
Subject	The subject of the mail.
ToRecipients	A string list of To recipients.

### 2.32.3 Methods

Methods name	Description
CanSendMail	Returns whether mail can be sent or not.

### 2.32.4 Public Properties

Property name	Description
MFMailComposeViewController	Returns a reference to the native iOS <code>MFMailComposeViewController</code> .

### 2.32.5 Published Events

Events name	Description
OnDidFinishWithResult	Event that is called when the mail is either sent, cancelled, saved or when an error occurred.

## 2.33 TMSFMXNativeMFMessageComposeViewController

---

### 2.33.1 Usage

The `MFMessageComposeViewController` class provides a standard system user interface for composing SMS (Short Message Service) text messages. Use this class to configure the initial recipients and body of the message, if desired, and to configure a delegate object to respond to the final result of the user's action—whether they chose to cancel or send the message.

### 2.33.2 Published Properties

Property name	Description
Body	The body of the message.
Recipients	A string list of the recipients of the message.

### 2.33.3 Public Properties

Property name	Description
CanSendText	Returns whether a message can be sent or not.
MFMessageComposeViewController	Returns a reference to the native iOS <code>MFMessageComposeViewController</code> .

### 2.33.4 Published Events

Events name	Description
OnDidFinishWithResult	Event called when the message is either sent, cancelled or when an error occurred.

## 2.34 TMSFMXNativeUIRichTextView

---

### 2.34.1 Usage

This component is based on the native iOS `UITextView` component and adds rich text editing capabilities. For more information about properties, methods and events that are not listed here please refer to the [TMSFMXNativeUITextView](#) component.

### 2.34.2 Published Properties

Property name	Description
ContextMenuOptions	When interacting with the TextView, a context menu pops up when selecting text. The context menus consists of various options such as cut, copy, and paste, select and select all editing capabilities. These menu items can be optionally disabled through this property;
DataDetectorTypes	This property can be used to specify the types of data (phone numbers, http links ...) that should be automatically converted to clickable URLs in the text view. When clicked, the text view opens the application responsible for handling the URL type and passes it the URL.

### 2.34.3 Public Properties


Property name	Description
Selection: TMSFMXNativeRichTextLibRange	Gets and Sets the selection on the TextView. Selection is a record of text character position and length of selection in number of characters.
DataText: string (supported from iOS 6)	Gets and Sets a compatible Archived XML String that can be used to persist the rich text contents of the TextView.

### 2.34.4 Public Methods

Each getter and setter of a specific attribute has optional parameters to apply the attribute value to text at a specific position and length inside the TextView. If the parameters are not specified, the value is applied to the selected text. Below is an example of setting a bold text:

```
//apply bold to the selected text
TMSFMXNativeUIRichTextView1.SetBold(True);

//apply bold to the text at position 5 with a length of 3
TMSFMXNativeUIRichTextView1.SetBold(True, 5, 3);
```

 **important notice**

Not all functionality that is listed below is supported on iOS versions earlier than iOS 7. Functionality that is only supported on iOS 7 and later is marked.

Property name	Description
AddBitmap(AValue: TBitmap; ALineHeight: Integer = -1; ALocation: Integer = -1);	Inserts a bitmap in the TextView. By default, the lineHeight is adapted to the bitmap height but can be overridden by setting <code>ALineHeight</code> parameter > -1. Also, by default, the bitmap is inserted at selection, unless the <code>ALocation</code> parameter is different than -1 and sets the insert character position.
AddBitmapFromFile(AValue: String; ALineHeight: integer -1; ALocation: Integer = -1);	Inserts a bitmap from file in the TextView. By default, the lineHeight is adapted to the bitmap height but can be overridden by setting <code>ALineHeight</code> parameter > -1. Also, by default, the bitmap is inserted at selection, unless the <code>ALocation</code> parameter is different than -1 and sets the insert character position.
CanRedo	Returns a Boolean whether the TextView can perform a Redo action.
CanUndo	Returns a Boolean whether the TextView can perform an Undo action.
Clear	Clears the text inside the TextView.
Copy	Copies the selected text on the clipboard.
Cut	Cuts the selected text on the clipboard.
CutAsPlainText	Cuts the selected text as plain text.
GetBackgroundColor / SetBackgroundColor	Gets or Sets the text background color.
GetBaselineOffset / SetBaselineOffset (iOS 7 and later only)	Gets or Sets the text baseline offset. The baseline offset is identical to subscript and superscript.
GetBold / SetBold	Gets or Sets the text bold.
GetFont / SetFont	Gets or Sets the text font name and size.
GetFontSize / SetFontSize	Gets or Sets the text font size.
GetForegroundColor / SetForegroundColor	Gets or Sets the text color.
GetItalic / SetItalic	Gets or Sets the text italic.
GetParagraphStyle / SetParagraphStyle	Gets or Sets the text paragraph style.
GetPlainText / GetPlainTextRange	Gets the plain text from the TextView, optionally specified by a text range.
GetRichText / GetRichTextRange	Gets the rich text from the TextView, optionally specified by a range.
Depending on the data type (see <a href="#">Import and Export chapter</a> )	

Property name	Description
GetStrikethrough / SetStrikethrough	Gets or Sets the text strikethrough style. The style can be a combination of a line style, pattern and / or grouped by word. The style is the same type used in the <code>GetUnderline</code> / <code>SetUnderline</code> functionality.
GetStrikethroughColor / SetStrikethroughColor (iOS 7 and later only)	Gets or Sets the text strikethrough color.
GetStrikethrough / SetStrikethrough	Gets or Sets the text strikethrough style. The style can be a combination of a line style, pattern and / or grouped by word. The style is the same type used in the <code>GetUnderline</code> / <code>SetUnderline</code> functionality.
GetStrikethroughColor / SetStrikethroughColor (iOS 7 and later only)	Gets or Sets the text strikethrough color.
GetStrokeColor / SetStrokeColor	Gets or Sets the text stroke color.
GetStrokeWidth / SetStrokeWidth	Gets or Sets the text stroke width.
GetSubscript / SetSubscript (iOS 7 and later only)	Gets or Sets the text subscript value offset. Can be combined with <code>SetFontSize</code> for a smaller font.
GetSuperscript / SetSuperscript (iOS 7 and later only)	Gets or Sets the text superscript value offset. Can be combined with <code>SetFontSize</code> for a smaller font.
GetTextLength	Returns the length of the text of a TextView.
GetUnderline / SetUnderline	Gets or Sets the text underline style. The style can be a combination of a line style, pattern and / or grouped by word.
GetUnderlineColor / SetUnderlineColor (iOS 7 and later only)	Gets or Sets the text underline color.
GetURL / SetURL (iOS 7 and later only)	Gets or Sets the text URL. The URL is only clickable when the TextView <code>Editable</code> property is set to false.
GetValues	Gets all values applied on the text.
Import / ExportData	Functionality to import / export the rich text from / to a file.
Depending on the data type (see <a href="#">Import and Export chapter</a> )	
ImportFromStream / ExportToStream	Functionality to import / export the rich text from / to a memory stream.
Depending on the data type (see <a href="#">Import and Export chapter</a> )	

Property name	Description
InitializeValues	Used to initialize the record with default values before passing it to the SetValues method.
Paste	Pastes the text from the clipboard.
PasteAsPlainText	Pastes the text from the clipboard as plain text.
Redo	Redoes the previous action (only available for plain text).
Select	Selects a specific range of text.
SelectAll	Selects all the text in the TextView.
SetRichText	Sets the rich text from a TextView, optionally specified by a range.
Depending on the data type (see <a href="#">Import and Export chapter</a> )	
ToggleBold	Toggles bold on the selected text.
ToggleItalic	Toggles italic on the selected text.
ToggleUnderline	Toggles underline on the selected text.
Undo	Undoes the last action (only available for plain text).

### 2.34.5 Import and export of (rich) text

The TextView supports importing and exporting the rich text to a stream, to a file and to a string. Some of the import / export functionality has the capability of adding an additional parameter to export to a plain, RTF, RTFD or HTML formatted data. The values of this parameter are listed below and specify which data type is supported on which iOS version and if import / export are supported:

Data Type	Supported iOS version	Import / Export
dtArchivedXMLDocumentType	iOS 6 and later	✓
dtPlainTextDocumentType	iOS 7 and later only	✓
dtRTFTextDocumentType	iOS 7 and later only	✓ (no image support)
dtRTFDTextDocumentType	iOS 7 and later only	✓
dtHTMLTextDocumentType	iOS 7 and later only	Export only



## 2.35 TMSFMXNativeUIRichTextViewToolbar

---

### 2.35.1 Usage

This component can be used separately or in combination with a `TMSFMXNativeUIRichTextView` component. The toolbar comes with a number of predefined actions for clipboard or changing attributes of rich text. This is represented as buttons on the toolbar. By default, all possible actions are visible on the toolbar but the `Options` property allows to customize this and hide specific actions. Set under `Options` the correct value to false to hide a specific action. When the `TMSFMXNativeUIRichTextViewToolBar` is connected to a `TMSFMXNativeUIRichTextView` (via assigning a `TMSFMXNativeUIRichTextView` to the `TMSFMXNativeUIRichTextViewToolBar.RichTextView` property), clicking on a toolbar action button will perform the clipboard action or apply the attribute value automatically to the selected text in the `TMSFMXNativeUIRichTextView`.

When no `TMSFMXNativeUIRichTextView` is connected to the `TMSFMXNativeUIRichTextViewToolBar`, the triggered event for the action can be used to programmatically apply the appropriate attribute.

## 2.36 TMSFMXNativeUIFontPicker

---

### 2.36.1 Usage

---

Based on the `TMSFMXNativeUIPopoverController` component and adds the supported fonts in a `PickerView`. The `SelectedFont` property can be used to preset the font and the `OnSelectFont` event is triggered when the value of the picker has changed.

## 2.37 TMSFMXNativeUIColorPicker

---

### 2.37.1 Usage

---

Based on the `TMSFMXNativeUIPopoverController` component and adds a set of colors in a `PickerView`. The `SelectedColor` property can be used to preset the color and the `OnSelectColor` event is triggered when the value of the picker has changed.

## 2.38 TMSFMXNativeMPMoviePlayerViewController



**important**

The `TMSFMXNativeMPMoviePlayerViewController` is deprecated, please use the `TMSFMXNativeAVPlayerViewController`

### 2.38.1 Usage

The `TMSFMXNativeMPMoviePlayerViewController` class implements a simple view controller for displaying movies.

### 2.38.2 Published Properties

Property name	Description
<code>AllowsAirPlay</code>	Property to determine if the current movie can be shown on airplay enabled devices or not.
<code>ControlStyle</code>	The style of the controls (previous, next, pause, play, stop, ...) depending on the view (embedded, fullscreen, ...).
<code>EndPlaybackTime</code>	The end time (measured in seconds) for playback of the movie.
<code>FullScreen</code>	Toggles between normal / embedded and fullscreen mode. <code>-1</code> by default which means the movie end time is used.
<code>InitialPlaybackTime</code>	The initial time (measure in seconds) for playback of the movie. <code>-1</code> by default which means the movie initial time is used.
<code>Location</code>	The location of the movie if the movie is a local file.
<code>RepeatMode</code>	Sets the repeat mode to none or one repeat.
<code>ScalingMode</code>	Sets the scaling mode of the movie to none, fill, aspect fill or aspect fit.
<code>ShouldAutoplay</code>	Property that determines whether the movie should automatically start playing.
<code>ShowInView</code>	Property that shows the movie inside the view rectangle, if false, use the public show method which displays the movie fullscreen as a separate view.
<code>SourceType</code>	The source type of the video, if known, the source type can be set to be a local movie file or a stream.
<code>URL</code>	The URL of the movie if the movie is an online movie.

### 2.38.3 Public Properties

Property name	Description
<code>MoviePlayerViewController</code>	Returns a reference to the native iOS <code>MPMoviePlayerViewController</code> .

## 2.38.4 Public Methods

Methods name	Description
AirPlayVideoActive: Boolean;	Returns whether the airplay video is active or not.
Duration: Double;	The duration of the movie.
Hide	Hide the movie if <code>ShowInView</code> is false.
NaturalSize: TSizeF;	The natural size of the current movie being played.
Pause	Pause the movie.
Play	Play the movie.
PlayableDuration: Double;	The playable duration of the movie. The amount of time in seconds of the already loaded content of the movie.
Playbackstate	The state of the movie, necessary to know if the video is able to be played.
ReadyForDisplay: Boolean;	Returns a Boolean if the movie is ready to be displayed.
Show	Show the movie fullscreen if <code>ShowInView</code> is false
Stop	Stop the movie.

## 2.38.5 Published Events

Event name	Description
OnDidEnterFullScreen	Event called when the movie is in fullscreen mode.
OnDidExitFullScreen	Event called when the movie is in normal / embedded mode.
OnPlaybackDidFinish	Event called when the movie stopped playing.
OnWillEnterFullScreen	Event called when the movie will enter fullscreen mode.
OnWillExitFullScreen	Event called when the movie will enter normal / embedded mode.

## 2.39 TMSFMXNativeUIActivityViewController

### 2.39.1 Usage

The `TMSFMXNativeUIActivityViewController` class is a standard view controller that you can use to offer various services from your application. The system provides several standard services, such as copying items to the pasteboard, posting content to social media sites, sending items via email or SMS, and more.

### 2.39.2 Published Properties

Property name	Description
ExcludedTypes	The types that are excluded from the <code>ActivityView</code> .

#### Public Properties

Property name	Description
<code>ActivityView</code>	Returns a reference to the native iOS <code>UIActivityViewController</code> .
<code>BitmapFiles</code>	List of strings of bitmaps that will be used in the <code>ActivityView</code> .
<code>Bitmaps</code>	List of <code>TBitmap</code> instances that will be used in the <code>ActivityView</code> .
<code>Values</code>	List of string values that will be used in the <code>ActivityView</code> .

### 2.39.3 Public Methods

Method name	Description
<code>Hide</code>	Hides the <code>ActivityView</code> .
<code>Show</code>	Shows the <code>ActivityView</code> .
<code>ShowFromButton(AButton: UIBarButtonItem)</code>	Shows the <code>ActivityView</code> from a <code>ToolBar</code> button. (iPad only)
<code>ShowFromControl(AControl: TMSFMXNativeUIBaseControl)</code>	Shows the <code>ActivityView</code> from a TMS FMX Native UI Control placed on the form. (iPad only)
<code>ShowFromRect(ARect: TRectF)</code>	Shows the <code>ActivityView</code> from a specific rectangle in the main view. (iPad only)
<code>ShowFromRectInView(ARect: TRectF; AView: UIView)</code>	Shows the <code>ActivityView</code> from a specific rectangle in a specific view. (iPad only)

## 2.40 TMSFMXNativeSLComposeViewController

---

### 2.40.1 Usage

The `TMSFMXNativeSLComposeViewController` class presents a view to the user to compose a post for supported social networking services.

### 2.40.2 Public Properties

Property name	Description
<code>SLComposeViewController</code>	Returns a reference to the native iOS <code>SLComposeViewController</code> .

### 2.40.3 Public Methods

Methods name	Description
<code>AddBitmap(ABitmap: TBitmap): Boolean;</code>	Adds a bitmap to the service.
<code>AddBitmapFile(ABitmapFile: String): Boolean;</code>	Adds a bitmap from a file to the service.
<code>AddMessage(AMessage: String): Boolean;</code>	Adds a message to the service.
<code>AddURL(AURL: String): Boolean;</code>	Adds an URL to the service.
<code>InitializeForServiceType(AServiceType: TTMSFMXNativeSLComposeViewControllerServiceType);</code>	Initializes the <code>SLComposeViewController</code> for the specific service.
<code>isServiceTypeAvailable(AServiceType: TTMSFMXNativeSLComposeViewControllerServiceType): Boolean;</code>	Shows a dialog to post to a specific service with a message, optional bitmap and url.
<code>Post(AServiceType: TTMSFMXNativeSLComposeViewControllerServiceType; AMessage: string; ABitmap: TBitmap; AURL: String): Boolean;</code>	Shows a dialog to post to a specific service with a message, optional bitmap and url.
<code>Post(AServiceType: TTMSFMXNativeSLComposeViewControllerServiceType; AMessage: string; ABitmapFile: String; AURL: String): Boolean;</code>	Shows a dialog to post to a specific service with a message, optional bitmap file and url.
<code>SLComposeViewController</code>	Returns a reference to the native iOS <code>SLComposeViewController</code> .

## 2.40.4 Published Events

---

<b>Events name</b>	<b>Description</b>
OnCancelled	Event called when the post action is cancelled.
OnDone	Event called when the post action is successful.

---



## 2.41 TMSFMXNativeUICollectionView

---

### 2.41.1 Overview

---

#### Usage

The `TMSFMXNativeUICollectionView` class manages an ordered collection of data items and presents them using customizable layouts. Collection views provide the same general function as table views except that a collection view is able to support more than just single-column layouts.

With this implementation, the visual representation is based on a template, which is available for the header, footer and the item. The `TMSFMXNativeUICollectionView` component comes with a designer that allows creating and modifying these templates. Each template allows adding various controls such as a label, image, button, stepper, etc... The template control is stored and accessible separately as a non-visual component.

## Methods

Methods name	Description
AddFooterTemplateButton	A set of overloads that adds a button template control to the Footer template.
AddFooterTemplateCheckBox	A set of overloads that adds a checkbox template control to the Footer template.
AddFooterTemplateControl	A set of overloads that allows adding a template control the the footer template. Additional properties need to be set to the template control result or parameter depending on the type of overload that is chosen.
AddFooterTemplateImageView	A set of overloads that adds an imageview template control to the Footer template.
AddFooterTemplateLabel	A set of overloads that adds a label template control to the Footer template.
AddFooterTemplateProgressView	A set of overloads that adds a progressview template control to the Footer template.
AddFooterTemplateStepper	A set of overloads that adds a stepper template control to the Footer template.
AddFooterTemplateSwitch	A set of overloads that adds a switch template control to the Footer template.
AddFooterTemplateTextField	A set of overloads that adds a textfield template control to the Footer template.
AddFooterTemplateTextView	A set of overloads that adds a textview template control to the Footer template.
AddHeaderTemplateButton	A set of overloads that adds a button template control to the Header template.
AddHeaderTemplateCheckBox	A set of overloads that adds a checkbox template control to the Header template.
AddHeaderTemplateControl	A set of overloads that allows adding a template control to the header template. Additional properties need to be set to the template control result or parameter depending on the type of overload that is chosen.
AddHeaderTemplateImageView	A set of overloads that adds an imageview template control to the Header template.
AddHeaderTemplateLabel	A set of overloads that adds a label template control to the Header template.
AddHeaderTemplateStepper	A set of overloads that adds a stepper template control to the Header template.
AddHeaderTemplateSwitch	A set of overloads that adds a switch template control to the Header template.

Methods name	Description
AddHeaderTemplateTextField	A set of overloads that adds a textfield template control to the Header template.
AddHeaderTemplateTextView	A set of overloads that adds a textview template control to the Header template.
AddItemTemplateButton	A set of overloads that adds a button template control to the item template.
AddItemTemplateCheckBox	A set of overloads that adds a checkbox template control to the item template.
AddItemTemplateControl	A set of overloads that allows adding a template control to the item template. Additional properties need to be set to the template control result or parameter depending on the type of overload that is chosen.
AddItemTemplateImageView	A set of overloads that adds an imageview template control to the item template.
AddItemTemplateLabel	A set of overloads that adds a label template control to the item template.
AddItemTemplateProgressView	A set of overloads that adds a progressview template control to the item template.
AddItemTemplateStepper	A set of overloads that adds a stepper template control to the item template.
AddItemTemplateSwitch	A set of overloads that adds a switch template control to the item template.
AddItemTemplateTextField	A set of overloads that adds a textfield template control to the item template.
AddItemTemplateTextView	A set of overloads that adds a textview template control to the item template.
DeselectItem / DeselectItems / DeselectAllItems	Deselects a specific or multiple item(s) identified with a section and row parameter.
GetFooterTemplateControl	Returns a footer template control with a specific section, row and id parameter.
GetHeaderTemplateControl	Returns a header template control with a specific section, row and id parameter.
GetItemTemplateControl	Returns an item template control with a specific section, row and id parameter.
ReloadData	Reloads the complete CollectionView, discarding and re-initializing items, headers and footers.
ReloadItem / ReloadItems	Visually update a single or an array of <code>TCollectionViewItem</code> with a section and row parameter.

<b>Methods name</b>	<b>Description</b>
ReloadSection / ReloadSections	Visually update a single or an array of Integer with a section parameter. Each reload of a section will also reload all items that the section holds.
ScrollToItem	Scrolls to a specific item with a section and row parameter. Optionally allows passing the scrollposition and whether scrolling needs to be performed with or without animation.
SelectedItems	Returns an array of selected items.
SelectItem / SelectItems / SelectAllItems	Selects and scrolls to a specific or multiple item(s) identified with a section and row parameter. Optionally allows passing the scrollposition and whether scrolling needs to be performed with or without animation.
VisibleItems	Returns an array of visible items.

## Public Events

### important note

Events that might require additional iOSApi units and have native iOS parameters, use only when the published events do not suffice

Events name	Description
OnAddItemBackground	Event called when the default background view is added, allowing for more customization.
OnAddItemSelectedBackground	Event called when the default selected background view is added, allow for more customization.
OnApplyFooterValues	Event called to allow more customization on the footer view after the template controls values are applied.
OnApplyHeaderValues	Event called to allow more customization on the Header view after the template controls values are applied.
OnApplyItemBackground	Event called to allow more customization on the background view. This event differs with the <a href="#">OnAddItemBackground</a> in a way that it is called when an item is being re-used instead of initialized. When the background remains static and does not change when scrolling, interacting with the items. The <a href="#">OnAddItemBackground</a> event needs to be used, else the <a href="#">OnApplyItemBackground</a> .
OnApplyItemSelectedBackground	Event called to allow more customization on the selected background view. This event differs with the <a href="#">OnAddItemSelectedBackground</a> in a way that it is called when an item is being re-used instead of initialized. When the selected background remains static and does not change when scrolling, interacting with the items. The <a href="#">OnAddItemSelectedBackground</a> event needs to be used, else the <a href="#">OnApplyItemSelectedBackground</a> .
OnApplyItemValues	Event called to allow more customization on the item view after the template controls values are applied.
OnInitializeFooterTemplate	Event called to allow more customization on the footer view after the template controls are created and added to the footer. This event is only called when the footer is created. Use this event if the changes you have made afterwards remain static else use the <a href="#">OnApplyFooterValues</a> event.
OnInitializeHeaderTemplate	Event called to allow more customization on the header view after the template controls are created and added to the header. This event is only called when the header is created. Use this event if the changes you have made afterwards remain static else use the <a href="#">OnApplyHeaderValues</a> event.
OnInitializeItemBackground	Event called with a native iOS <a href="#">UIView</a> parameter that represents the item view and optionally allows creating an item background view that is used to apply the <a href="#">Options.ItemBackgroundColor</a> property. Setting the <a href="#">ACreate</a> parameter to false allows you to create your own native background view.
OnInitializeItemSelectedBackground	Event called with a native iOS <a href="#">UIView</a> parameter that represents the item view and optionally allows creating an item selected background view that is used to apply the <a href="#">Options.ItemSelectedBackgroundColor</a> property. Setting the <a href="#">ACreate</a> parameter to false allows you to create your own native selected background view.

Events name	Description
OnInitializeItemTemplate	Event called to allow more customization on the item view after the template controls are created and added to the item. This event is only called when the item is created. Use this event if the changes you have made afterwards remain static, else use the <a href="#">OnApplyItemValues</a> event.

---

### Published Events

The most important events are listed below, events such as an `OnItemButtonClick`, `OnItemStepperValueChanged` and equivalents for `Header` and `Footer` are not listed. These events depend on the kind of template that is constructed and used



at runtime. The CollectionView is implemented with a virtual mode and requires some of the events that are listed below (marked in red).

Events name	Description
OnGetItemBackgroundColor	Event called to retrieve the background color of an item.
OnGetItemSelectedBackgroundColor	Event called to retrieve the background color of an item used in selected state.
OnGetInsetForSectionAtIndex	Event called to retrieve the inset for a section at a specific index.
OnGetMinimumInteritemSpacingForSectionAtIndex	Event called to retrieve the minimum spacing between items on the same line in a section.
OnGetMinimumLineSpacingForSectionAtIndex	Event called to retrieve the minimum spacing between lines in a section.
OnGetReferenceSizeForFooterInSection	Event called to retrieve the reference size of the footer in a specific section.
OnGetReferenceSizeForHeaderInSection	Event called to retrieve the reference size of the header in a specific section.
OnGetSizeForItemAtIndexPath	Event called to retrieve the size for an item at a specific section and row.
OnDidSelectItem	Event called when an item is selected.
OnDidDeselectItem	Event called when an item is deselected.
OnShouldSelectItem	Event called before an item will be selected. You can use this event to prevent selection of specific items.
OnShouldDeselectItem	Event called before an item will be deselected. You can use this event to prevent deselection of specific items.
<b>OnGetNumberOfSections</b>	Event called to retrieve the number of sections in the UICollectionView. When this method is not implemented, the UICollectionView returns <b>1</b> section by default.
<b>OnGetNumberOfItemsInSection</b>	Event called to retrieve the number of items in the UICollectionView. When this method is not implemented, the UICollectionView returns <b>5</b> items per section by default.
<b>OnAddHeaderControl</b>	Event called when a header template control is initialized. Implement this event to set properties on a specific template control that remains static during the usage of the UICollectionView.
<b>OnAddFooterControl</b>	Event called when a footer template control is initialized. Implement this event to set properties on a specific template control that remains static during the usage of the UICollectionView.

Events name	Description
<a href="#">OnAddItemControl</a>	Event called when an item template control is initialized. Implement this event to set properties on a specific template control that remains static during the usage of the <code>CollectionView</code> .
<a href="#">OnApplyHeaderValue</a>	Event called when a header template control will apply its values. Implement this event to set properties on a specific template control that is dynamic and changes its values depending on section parameter.
<a href="#">OnApplyFooterValue</a>	Event called when a footer template control will apply its values. Implement this event to set properties on a specific template control that is dynamic and changes its values depending on the section parameter.
<a href="#">OnApplyItemValue</a>	Event called when an item template control will apply its values. Implement this event to set properties on a specific template control that is dynamic and changes its values depending on the section and row parameter.

## Templates

The `CollectionView` core is based on templates. Each section in the `CollectionView` consists of a header, footer and a number of items (elements). During creation, these elements are initialized and reused where possible, the `CollectionView` reads the controls inside the template, creates them and triggers a set of events (explained in the table of public and published events) that can be used to assign a value to a specific control inside an element based on the section and row parameter.

For header and footers, the row parameter will always be `0`.

On `CollectionView` level, the template used to visualize an element is stored inside a generic list of `TTMSFMXNativeUICollectionViewController`. This class has a number of descendants that define a native iOS control that can be used inside an element.

## First Initialization

Before the `CollectionView` will start displaying sections, with optional header and footer, the `CollectionView` needs `2` events implemented. The first one is the `OnGetNumberOfSections` which returns `1` by default, if the event is not implemented. The second one is the `OnGetNumberOfItemsInSection` which returns `5` by default and needs to be mapped to the number of items / sections in your data structure. Afterwards, you can start adding template controls to visualize your data per item and / or section. In the code sample below, we add `2` sections, and respectively `5` and `3` items.

```
procedure TForm1.TMSFMXNativeUICollectionViewController1GetNumberOfItemsInSection(
  Sender: TObject; ASection: Integer; var ANumberOfItems: Integer);
begin
  if ASection = 0 then
    ANumberOfItems := 5
  else
    ANumberOfItems := 3;
end;
```

```

procedure TForm1.TMSFMXNativeUICollectionV1GetNumberOfSections(
  Sender: TObject; var ANumberOfSections: Integer);
begin
  ANumberOfSections := 2;
end;

```

## Adding template controls

Programmatically adding a control to the template of an element can be done with one of the various functions that are listed in the table above. Each element has its own template and thus its own set of functions. For an item the function starts with "AddItemTemplate", for a header and footer respectively "AddHeaderTemplate" and "AddFooterTemplate". The second part is based on the type of template control you wish to add to the element. Below is a sample of adding a label to the item template.

```

procedure TForm1.FormCreate(Sender: TObject);
var
  lbl: TTMSFMXNativeUICollectionV1TemplateLabel;
begin
  lbl := TMSFMXNativeUICollectionV1.AddItemTemplateLabel(10, 10, 100, 25, 'Hello World !');
end;

```

The code adds a label template control descendant to the item template collection that is located under the `Template` property on `CollectionView` level. As explained in the previous chapter, the `CollectionView` then loads and initializes the header, footer and item elements and loops through the template collection of each element. In this sample, a native iOS `UILabel` will be added to the item.

## Initializing / modifying values

After defining the template for either the header, footer and / or the item, the `CollectionView` initializes the elements and creates a native iOS control for each template control. The `CollectionView` manages each element separately in memory and reuses its elements where possible.

The `CollectionView` does not manage a separate data structure, so the data that needs to be visualized will need to be passed by implementing some events. The "First Initialization" chapter already mentioned events to determine how many sections and items are going to be displayed. Next, after adding template controls to the header, footer and / or item with optionally default values, the data can be mapped on the control by implementing `2` events. These events are also called per element which means that we have an equivalent for the header, footer and item. In this sample code, we continue with our label template control that we have programmatically added.

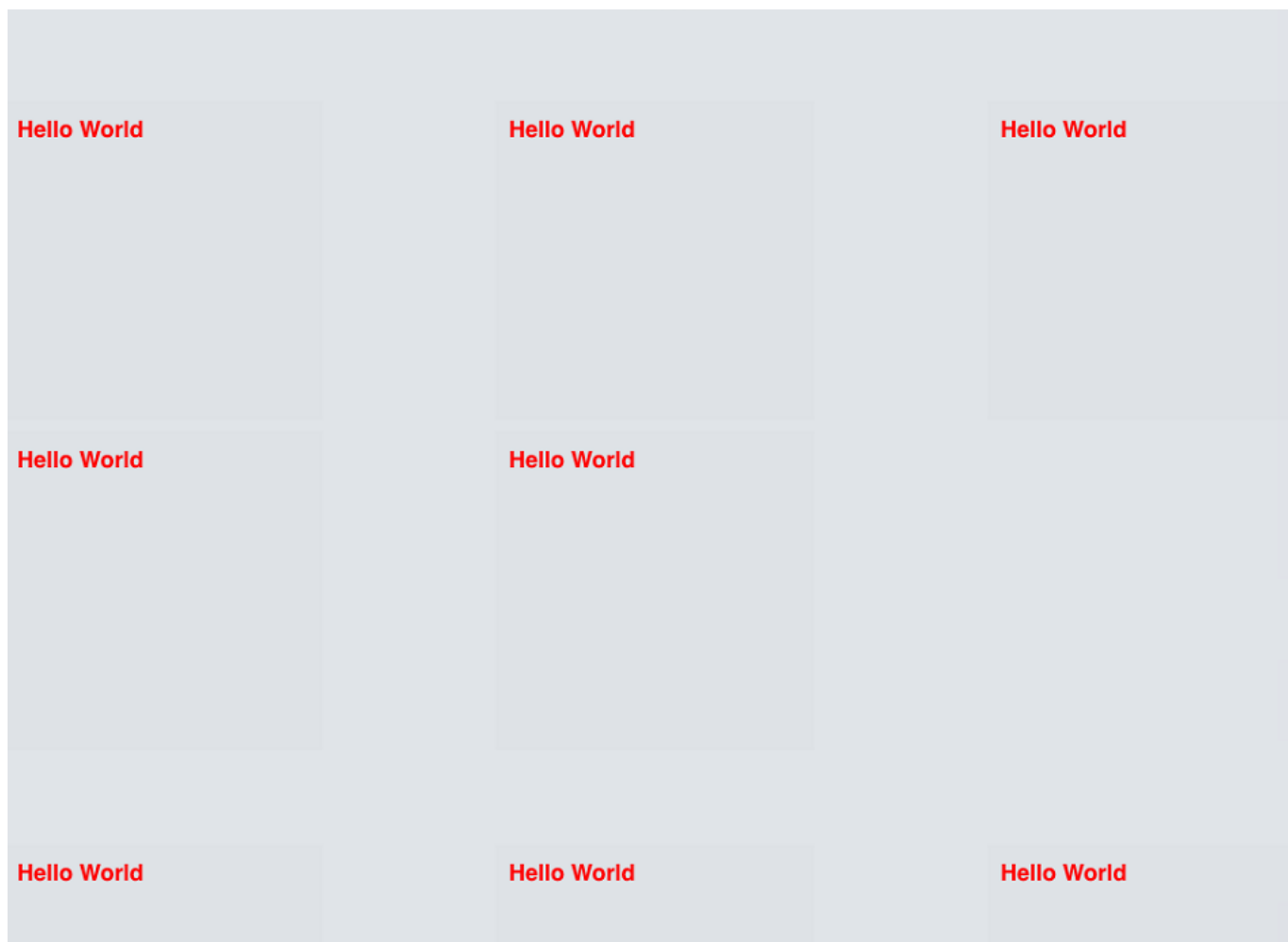
The code shows how to define a default font, font size and text color that will be applied to all items.

```

procedure TForm1.FormCreate(Sender: TObject);
var
  lbl: TTMSFMXNativeUICollectionV1TemplateLabel;
begin
  lbl := TMSFMXNativeUICollectionV1.AddItemTemplateLabel(10, 10, 100, 25, 'Hello World !');
  lbl.Font.Name := 'Helvetica Bold';
  lbl.Font.Size := 18;
  lbl.TextColor := TAlphaColorRec.Red;
end;

```

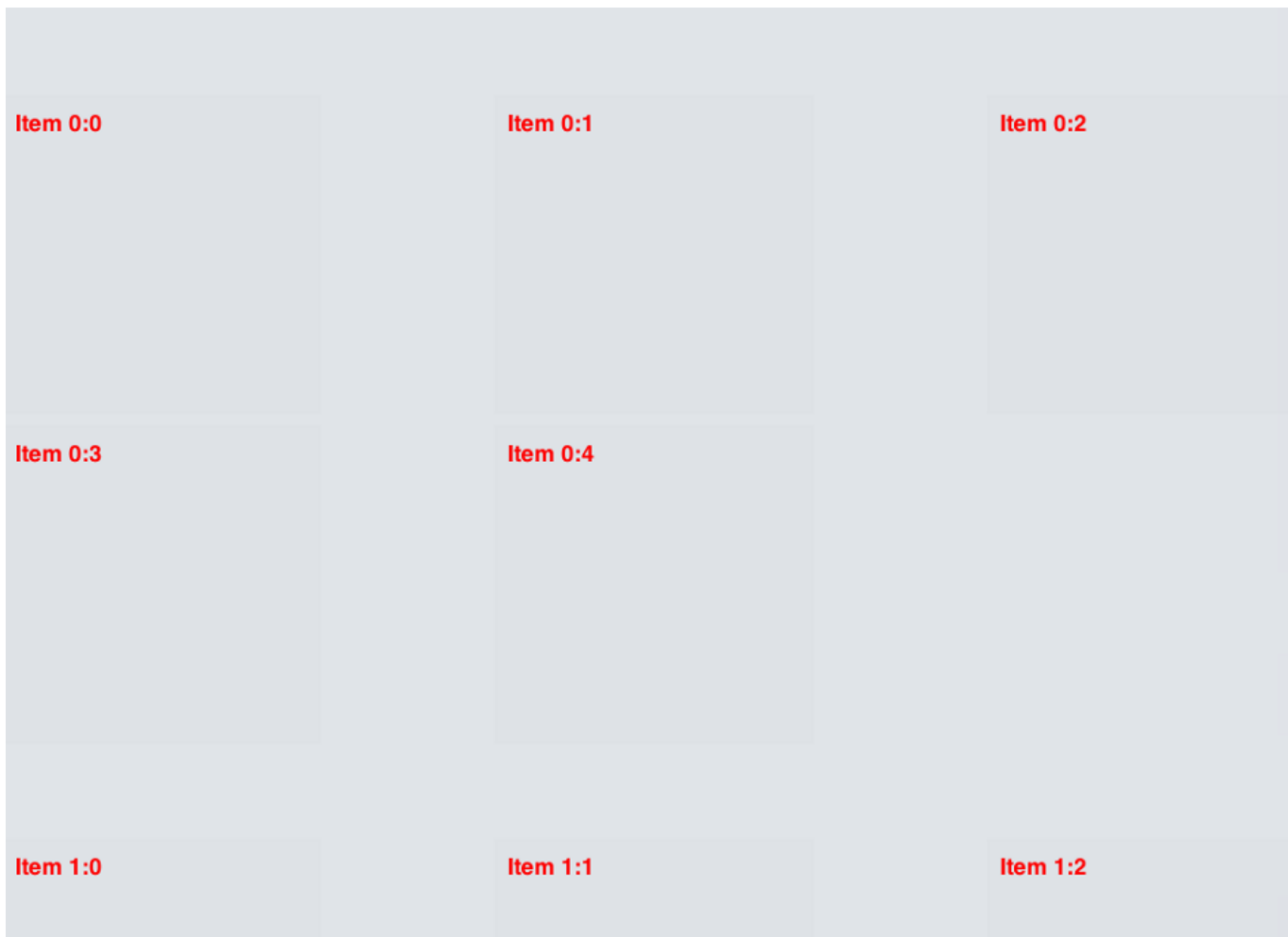
Running the application will show a `CollectionView` with `5` items per section, with red bold text and the default value "Hello World" that is set in the `FormCreate`.



As explained, each label has the same color, font and text based on the template control of the item element. To modify these default values dynamically, we will need to implement an event that manages this. The `OnApplyItemValue` event is called with a section and row parameter and allows you to modify the value of the item. There are also equivalents for the header and footer that will be demonstrated after adding a template control to the header as well.

The code below shows how to access the label with a special helper function that is available in the `FMX.TMSNativeUICollectionView` unit to detect if the template control is a `Label` control. Each control inherits from the base control and is passed as a parameter of this event.

```
procedure TForm1.TMSFMXNativeUICollectionView1ApplyItemValue(Sender: TObject;
  AControl: TMSFMXNativeUICollectionViewTemplateControl; ASection,
  ARow: Integer);
begin
  if IsLabel(AControl) then
    AsLabel(AControl).Text := 'Item ' + inttostr(ASection) + ':' + inttostr(ARow);
end;
```



Now, the header is visible as well, but does not show a template control, so we can use the same approach as like we did with the item, and add a control to the header template and modify its value afterwards through the appropriate event. The same could be done for the footer, but then you would need to set the `FooterVisible` property to true which is located under the `Options` property. The complete code for this sample can be found below:

```

procedure TForm1.FormCreate(Sender: TObject);
var
  lbl: TTMSFMXNativeUICollectionViewTemplateLabel;
begin
  lbl := TMSFMXNativeUICollectionView1.AddItemTemplateLabel(10, 10, 100, 25, 'Hello World !');
  lbl.Font.Name := 'Helvetica Bold';
  lbl.Font.Size := 18;
  lbl.TextColor := TAlphaColorRec.Red;

  lbl := TMSFMXNativeUICollectionView1.AddHeaderTemplateLabel(10, 10, 100, 25, '');
  lbl.Font.Name := 'Helvetica Bold';
  lbl.Font.Size := 22;
  lbl.TextColor := TAlphaColorRec.Blue;
end;

procedure TForm1.TMSFMXNativeUICollectionView1ApplyHeaderValue(
  Sender: TObject; AControl: TTMSFMXNativeUICollectionViewTemplateControl;
  ASection, ARow: Integer);
begin
  if IsLabel(AControl) then
    AsLabel(AControl).Text := 'Section ' + inttostr(ASection);
end;

```

```

end;

procedure TForm1.TMSFMXNativeUICollectionVew1ApplyItemValue(Sender: TObject;
  AControl: TMSFMXNativeUICollectionViewTemplateControl; ASection,
  ARow: Integer);
begin
  if IsLabel(AControl) then
    AsLabel(AControl).Text := 'Item ' + inttostr(ASection) + ':' + inttostr(ARow);
end;

procedure TForm1.TMSFMXNativeUICollectionVew1GetNumberOfItemsInSection(
  Sender: TObject; ASection: Integer; var ANumberOfItems: Integer);
begin
  if ASection = 0 then
    ANumberOfItems := 5
  else
    ANumberOfItems := 3;
end;

procedure TForm1.TMSFMXNativeUICollectionVew1GetNumberOfSections(
  Sender: TObject; var ANumberOfSections: Integer);
begin
  ANumberOfSections := 2;
end;

```

## Section 0

Item 0:0

Item 0:1

Item 0:2

Item 0:3

Item 0:4

## Section 1

Item 1:0

Item 1:1

Item 1:2

## Identifiers

In the sample above, we have added one label to the header and one to the item template. By default each template control is uniquely identified by its index. If you wish to add an additional template label to the item, the identifier is incremented and accessible in the order that they are added to the template. The identifier starts from `1` for the first item. The code of the `OnApplyItemValue` modified with a “description label” could then be implemented like the sample below:

```
procedure TForm1.TMSFMXNativeUICollectionView1ApplyItemValue(Sender: TObject;
  AControl: TTMSFMXNativeUICollectionViewTemplateControl; ASection,
  ARow: Integer);
begin
  if IsLabel(AControl) and (AControl.GetViewID = 1) then
    AsLabel(AControl).Text := 'Title ' + inttostr(ASection) + ':' + inttostr(ARow);

  if IsLabel(AControl) and (AControl.GetViewID = 2) then
    AsLabel(AControl).Text := 'Description ' + inttostr(ASection) + ':' + inttostr(ARow);
end;
```

If you have a special template control that needs to be accessible in an easy way, you can specify an ID to your template control. The `ID` property needs to be `1` or higher and is returned by the `GetViewID` function to identify the correct control. If you have multiple template controls of the same type and want to manage them with an easier accessibility, this is the way to identify them.

## Interaction

The `CollectionView` supports a variety of template controls such as a label, imageview, progressview but also interactable controls such as a button. Adding a button is done in the same way as adding a label, can optionally and uniquely be identified and is also accessible to modify its values dynamically.

The `CollectionView` exposes events that are triggered when clicking a button, editing text in a textview or toggling the switch, with a section and row parameter.

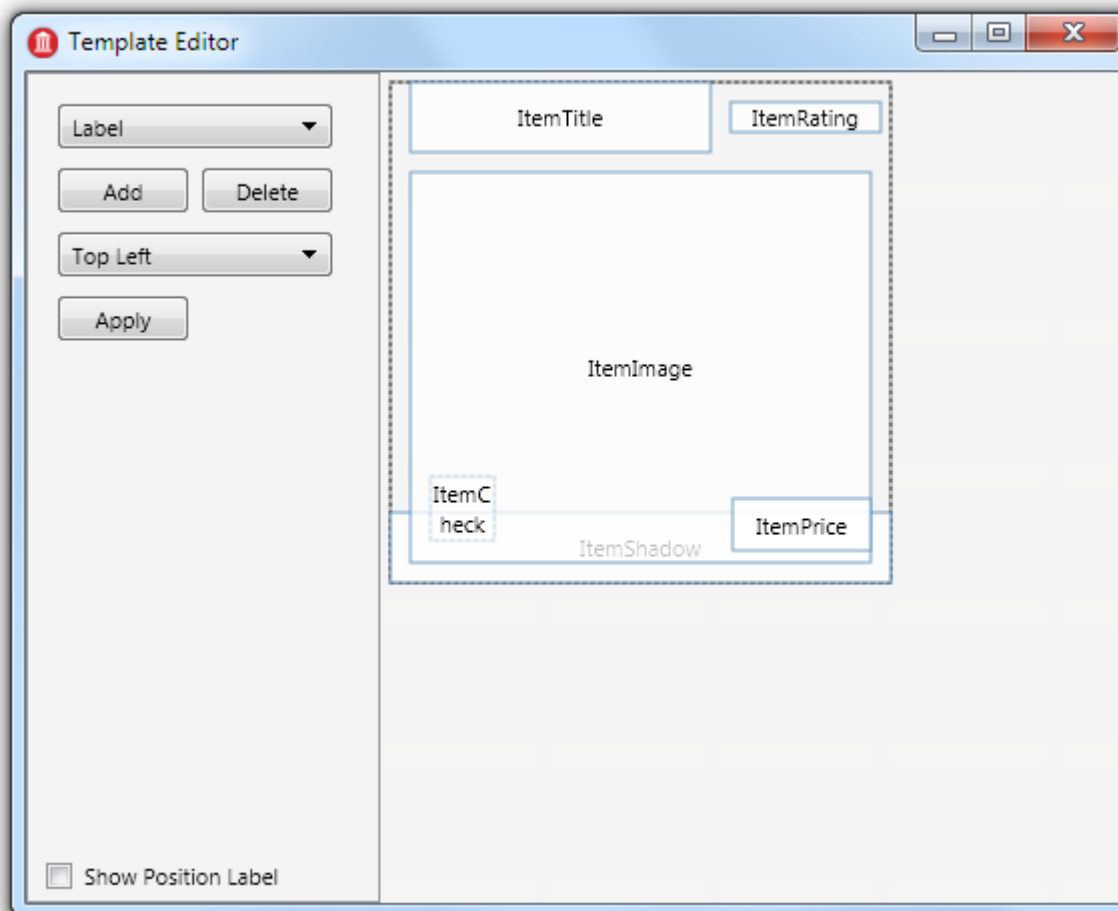
## Designtime editor

Programmatically adding, modifying and positioning controls can be useful to quickly have a rough idea on how your application would possibly look like. You can then concentrate on the data structure, implement interaction, selection, database handling, etc.... and handle the visualization afterwards. It might also be sufficient if you only have a label and an image and they are positioned without indenting, under each other with the image taking up the remaining space after subtracting the label height from the item height, but this will be easier said than done in some cases.

To cover the “easier said than done” part, we have created a designtime template editor that acts as a helper to create a template for the header, footer and / or item element. Under the `Template` property on `CollectionView` level you will notice three templates. Click on the three dots next to the template of choice to start the editor.

The editor will display a dotted rectangle that represents the boundaries of the element depending on the chosen template. On the left you can add template controls, position them relatively inside the element rectangle. A newly created template control is accessible as a non-visual component both at designtime and at runtime. It can also be removed as easily as it has been added. Below is the template of the item element shown inside the editor, and the result after the application is started and the data is loaded.





## Performance

The CollectionView manages a state where it checks if a property is modified, sets a flag and updates the appropriate properties on the template control in one of the `OnApply*Value` events, but each control that is added to the template is passed as a parameter to this event so you can manipulate the data and visual aspects. When having multiple controls that

remain identical for each element during the lifetime of the `CollectionView`, such as image shadows, text, buttons it is unnecessary to call the `OnApply*Value` event for these template controls. This will affect performance in a positive way and will allow you to manage the controls that really matter for your application.

When adding a new template control, you will notice a `State` property that is being set to `csDynamic` by default. In the demo you will notice by selecting the shadow item template control that the `State` property is being set to `csStatic` as the shadow is identical for each item.

## 2.41.2 Properties

---

### Overview

#### PUBLISHED PROPERTIES

Property name	Description
<a href="#">Options</a>	Options to configure the <code>CollectionView</code> .
<a href="#">Template</a>	A set of templates used to visualize various elements inside the header, footer or item.

---

#### PUBLIC PROPERTIES

Property name	Description
<code>CollectionView</code>	Returns a reference to the native iOS <code>UICollectionView</code> .
<code>CollectionViewController</code>	Returns a reference to the native iOS <code>UICollectionViewController</code> .
<code>CollectionViewFlowLayout</code>	Returns a reference to the native iOS <code>UICollectionViewFlowLayout</code> .

---

## Options

## OVERVIEW

Property name	Description
AllowsMultipleSelection	Enables or disables multi selection in the CollectionView.
AllowsSelection	Enables or disables selection in the CollectionView.
BackgroundColor	The backgroundColor of the CollectionView. When the value is set to <code>TAlphaColorRec.Null</code> , the background is transparent allowing additional views to be visible when placed behind the CollectionView, such as an ImageView used as a texture background
FooterReferenceHeight	The height of the footer, when the footer is visible in vertical scrolling mode. The width of the footer is stretched depending on the width of the control and the section insets. The height can be customized per section through an additional event.
FooterReferenceWidth	The width of the footer, when the footer is visible in horizontal scrolling mode. The height of the footer is stretched depending on the height of the control and the section insets. The width can be customized per section through an additional event.
FooterVisible	Toggles the footer visibility
HeaderReferenceHeight	The height of the header, when the header is visible in vertical scrolling mode. The width of the header is stretched depending on the width of the control and the section insets. The height can be customized per section through an additional event.
HeaderReferenceWidth	The width of the header, when the header is visible in horizontal scrolling mode. The height of the header is stretched depending on the height of the control and the section insets. The width can be customized per section through an additional event.
HeaderVisible	Toggles the header visibility.
ItemBackgroundColor	The background color of the item, which is transparent by default ( <code>TAlphaColorRec.Null</code> ).
ItemHeight	The default height of the item. The height can be customized per item through an additional event.
ItemSelectedBackgroundColor	The background color of the item in selected state.
ItemWidth	The default width of the item. The width can be customized per item through an additional event.
MinimumInteritemSpacing	Minimum spacing between items on the same line in a section. Can be customized per section through an additional event.
MinimumLineSpacing	Minimum spacing between lines in a section. Can be customized per section through an additional event.

## Scrolling

**Property name****Description**

Additional scrolling related properties, for more information on properties that are not listed here, please refer to the [TMSFMXNativeUIScrollView](#) component.

---

[Go back to Properties](#)

## SCROLLING

Property name	Description
Direction	The direction in which the CollectionView needs to scroll. Based on this direction, the header, footer and items are positioned and animated differently.

---

[Go back to Options](#)



## Template

Property name	Description
FooterTemplate	A collection of footer template controls, accessible and editable at designtime through the object inspector and the separate designer form.
HeaderTemplate	A collection of header template controls, accessible and editable at designtime through the object inspector and the separate designer form.
ItemTemplate	A collection of item template controls, accessible and editable at designtime through the object inspector and the separate designer form.

[Go back to Options](#)

## 2.42 TMSFMXNativeUIActivityIndicatorView

---

### 2.42.1 Usage

Use an activity indicator to show that a task is in progress. An activity indicator appears as a “gear” that is either spinning or stopped.

### 2.42.2 Published Properties

---

Properties name	Description
Color	The color of the indicator.
HidesWhenStopped	Hides the indicator when animation is stopped.
Style	The style of the indicator.

### 2.42.3 Public Methods

---

Methods name	Description
Indicator	Returns a reference to the native iOS <code>UIActivityIndicatorView</code> .
StartAnimating	Starts animating the indicator.
StopAnimating	Stops animating the indicator.
IsAnimating	Returns a <code>Boolean</code> whether the indicator is animating.

## 2.43 TMSFMXNativeUIWebView

---

### 2.43.1 Usage

You use the `TMSFMXNativeUIWebView` class to embed web content in your application.

### 2.43.2 Published Properties

Properties name	Description
ScalesPageToFit	Scales the page to fit the size of the WebView.

### 2.43.3 Public Properties

Properties name	Description
WebView	Returns a reference to the native iOS <code>UIWebView</code> .

### 2.43.4 Public Methods

Methods name	Description
CanGoBack: Boolean;	Returns a <code>Boolean</code> if the WebView can go back.
CanGoForward: Boolean;	Returns a <code>Boolean</code> if the WebView can go forward.
ExecuteJavaScript(AScript: String): String;	Executes Javascript on the current page.
GoBack;	Goes back one page in the WebView.
GoForward;	Goes forward one page in the WebView.
isLoading: Boolean;	Returns a <code>Boolean</code> whether the WebView is loading or not.
LoadFile(AFile: String);	Loads a specific file inside the WebView.
LoadHTMLString(AHTML: String);	Loads a specific HTML string or HTML content inside the WebView.
Navigate(AURL: String);	Navigates to a specific URL.
Reload;	Reloads the current page.
StopLoading;	Stops loading the current page.

## 2.43.5 Published Events

---

Events name	Description
OnDidFailLoadWithError	Event called when the loading failed.
OnDidFinishLoad	Event called when the loading is finished.
OnDidStartLoad	Event called when the loading started.
OnShouldStartLoadWithRequest	Event called when the WebView should start loading with a specific request.

## 2.43.6 Executing Javascript

---

The WebView has a function `ExecuteJavaScript` that executes javascript code on the current page. The following code will show an alert dialog with a “Hello World” message:

```
TMSFMXNativeUIWebView1.ExecuteJavaScript('alert("Hello World");');
```

## 2.43.7 Loading HTML

---

Other than loading a page through an URL, the WebView can also display HTML from a string. The `LoadHTMLString` functionality loads the HTML tags / content and displays the HTML inside the WebView.

## 2.44 TMSFMXNativeiCloud

---

### 2.44.1 Usage

The `TMSFMXNativeiCloud` component is used to access the iCloud key-value store. You typically use this component to make preference, configuration, and app-state data available to every instance of your app on every device connected to a user's iCloud account. More information about the iCloud key-value store can be found on the following page:

<https://developer.apple.com/library/mac/documentation/General/Conceptual/iCloudDesignGuide/Chapters/DesigningForKey-ValueDataIn iCloud.html>

### 2.44.2 Methods

Methods name	Description
AddKey	Adds a new key with a specific name and value to the iCloud key-value store.
KeyByName	Retrieves the key from the key collection after the keys have been loaded from the iCloud key-value store.
KeyValues[AKeyName]	Accesses the key value after the keys are loaded from the iCloud key-value store.
RegisterForKeyUpdates	Enabled by default through the <code>AutoSynchronize</code> property. Can be used to register the application to the notification center to receive iCloud key-value store updates.
RemoveAllKeys	Removes all the keys from the iCloud key-value store.
RemoveKey	Removes a specific key from the iCloud key-value store.
RemoveKeyByName	Removes a specific key from the iCloud key-value store based on the name.
SynchronizeKeys	Starts an asynchronous synchronize operation to retrieve the changed, keys from the iCloud key-value store.
UnRegisterForKeyUpdates	Used to unregister the application and no longer receive iCloud key-value store updates. The updates can be fetched manually by calling <code>UpdateKeys</code> .
UpdateKeys	Forces a synchronize operation and retrieves all keys from the iCloud key value store.

### 2.44.3 Properties

Properties name	Description
AutoSynchronize	Turn the automatic synchronization of keys on or off.
Keys	Public access to the key collection synchronized with the iCloud key-value store.

## 2.44.4 Events

Events name	Description
OnAccountChanged	Event called when the iCloud account changed on the user device.
OnKeyAdded	Event called when a key has been added from another location.
OnKeyRemoved	Event called when a key has been removed from another location.
OnKeysChanged	Event called when the key collection has changed.
OnKeyUpdate	Event called when a key store in the collection has changed.
OnQuotaViolation	Event called when the total available key-value store size has been exceeded. The key(s) that exceed this limited size will not be added to the iCloud key-value store.

## 2.44.5 Supported types

The `TMSFMXNativeiCloud` component keeps the iCloud keys synchronized (optionally with the `AutoSynchronize` property) with the key-value store. Each key has a `Value` property of type `TValue`. The supported types are `Integer`, `Double`, `Boolean`, `String` and `TMemoryStream`. There are multiple ways of persisting and retrieving the data. The methods and functions that can be used to perform this task are listed in the above table.

## 2.44.6 Entitlements

Before iCloud can be used in your application you need to enable it and sign your application. Additional information about enabling iCloud and incorporating it into your application can be found on the following page:

<https://developer.apple.com/library/ios/documentation/General/Conceptual/iCloudDesignGuide/Chapters/iCloudFundamentals.html>

After reading the guide, you will need to perform 2 steps: signing your device, and adding an entitlements file that adds the necessary keys to gain access to the iCloud storage. As a helper sample we have included an iCloud demo project that demonstrates how the Entitlements file is added.

When opening the Entitlements file (`iCloud.entitlements`) you will notice placeholders that need to be filled in with a combination of the Team-ID and the Bundle Identifier

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>get-task-allow</key>
  <true/>
  <key>com.apple.developer.ubiquity-container-identifiers</key>
  <array>
    <string>$(TeamIdentifierPrefix)com.mycompany.myapplication</string>
  </array>
  <key>com.apple.developer.ubiquity-kvstore-identifier</key>
  <string>$(TeamIdentifierPrefix)com.mycompany.myapplication</string>
</dict>
</plist>
```

The first key is to allow the debugger to access the application. This is inherited from the default entitlements file that is distributed when deploying your application. The `com.apple.developer.ubiquity-container-identifiers` and the `com.apple.developer.ubiquity-kvstore-identifier` keys are used to access iCloud. Here you need to specify the correct Team Identifier Prefix and the Bundle Identifier that matches your Application ID, used in the generation of the provisioning profile. Below is a sample of the Application ID at [developer.apple.com](https://developer.apple.com), used to generate a provisioning profile to sign your application.

FireMonkeySample
com.tmssoftware.FireMonkeySample

ID

**Name:** FireMonkeySample

**Prefix:** ABC123

**ID:** com.tmssoftware.FireMonkeySample

---

**Application Services:**

Service	Development	Distribution
<b>Data Protection</b>	<input type="radio"/> Disabled	<input type="radio"/> Disabled
<b>Game Center</b>	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
<b>iCloud</b>	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
<b>In-App Purchase</b>	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
<b>Inter-App Audio</b>	<input type="radio"/> Disabled	<input type="radio"/> Disabled
<b>Passbook</b>	<input type="radio"/> Disabled	<input type="radio"/> Disabled
<b>Push Notifications</b>	<input checked="" type="radio"/> Configurable	<input checked="" type="radio"/> Configurable

If the prefix is `ABC123` and the ID is `com.tmssoftware.FireMonkeySample`. The correct Entitlements.plist file would have `ABC123.com.tmssoftware.FireMonkeySample` as substitute for `$(TeamIdentifierPrefix)com.mycompany.myapplication`.

## 2.45 TMSFMXNativeiCloudDocument

---

### 2.45.1 Usage

The `TMSFMXNativeiCloudDocument` component is used to access the iCloud document storage. You typically use this component to add and update existing or create new documents and make them available to every instance of your app on every device connected to a user's iCloud account. More information about the iCloud document storage can be found on the following page:

<https://developer.apple.com/library/mac/documentation/General/Conceptual/iCloudDesignGuide/Chapters/DesigningForDocumentsIniCloud.html>

### 2.45.2 Properties

Properties name	Description
ContainerIdentifier	Optional property to specify a different container identifier to access your documents, such as the difference between a trial and a paid application.

---



## 2.45.3 Methods

Methods name	Description
AddDocument	Adds a new document to the Documents collection and moves the file to iCloud. When the file is added, the <code>OnDocumentAdded</code> event is called.
DeleteDocument	Deletes an existing document from iCloud and removes the entry from the collection.
DocumentByIndex	Returns the document by the index in the Documents collection.
DocumentByName	Returns the document by the file system name or the display name. These are properties that are extracted from the file as metadata when the documents are loaded.
DocumentCount	Returns the number of documents in the collection.
LoadDocuments	Loads the documents from iCloud. The <code>LoadDocuments</code> is the first step you need to manually implement after iCloud has been initialized. The <code>OnInitialized</code> event is triggered when iCloud has been loaded, or has failed to load. In this event, you need to call this method to asynchronously load the documents.
RefreshDocuments	<p>Manually refresh the documents asynchronously. When the documents are refreshed, the <code>OnDocumentsRefreshed</code> event is called. This event is also called when there are changes in the iCloud document storage.</p> <p>Each refresh automatically calls <code>OnDocumentAdded</code>, <code>OnDocumentDeleted</code> and <code>OnDocumentUpdated</code> based on the difference of the current and the previous documents state. The Documents collection is automatically updated.</p>
RemoveDocument	Removes the document from the collection and moves an existing document from iCloud to a local directory.
SwitchContainer	Switches between containers, after the <code>ContainerIdentifier</code> has been set. The currently loaded documents are cleared and renewed with the documents in the other container. If the <code>ContainerIdentifier</code> is an empty string, the default container is loaded, specified in your entitlements file.
UpdateDocument	Updates an existing document, this call has a number of overloads to update a document from a file or directly from a memory stream.

## 2.45.4 Events

Events name	Description
OnDocumentAdded	Event called when a new document is added to iCloud.
OnDocumentDeleted	Event called when an existing document is deleted from iCloud.
OnDocumentRemoved	Event called when an existing document is moved from iCloud to a local directory.
OnDocumentSaved	Event called when an existing document is updated and saved.
OnDocumentUpdated	Event called when an existing document is updated from iCloud.
OnDocumentsLoaded	Event called when the documents are loaded, after calling <code>LoadDocuments</code> .
OnDocumentsRefreshed	Event called when the documents are refreshed.
OnInitialized	Event called when iCloud is initialized.
OnDocumentDataChanged	Event called when an iCloud document data has changed.

## 2.45.5 Initialization

When dropping a component on the form, it will try to connect to the iCloud document storage container specified by the `ContainerIdentifier` property. If you have no intention to create multiple containers (such as the difference between a paid and a trial application), leave the `ContainerIdentifier` empty, so the default container is accessed. As this process is asynchronous, an event is triggered when the component is done initializing.

After initialization succeeds, the documents can be loaded. If the initialization fails, you can try to reconnect by calling

```
TMSFMXNativeiCloudDocument1.SwitchContainer;
```

```
procedure TForm1.TMSFMXNativeiCloudDocument1Initialized(Sender: TObject;
  ASuccess: Boolean);
begin
  if ASuccess then
    TMSFMXNativeiCloudDocument1.LoadDocuments;
end;
```

When the documents are loaded, the `OnDocumentsLoaded` event is called, and the listbox can be filled with the names of the documents.

```
procedure TForm1.TMSFMXNativeiCloudDocument1DocumentsLoaded(Sender: TObject);
var
  doc: TTMSFMXNativeiCloudDocumentItem;
  I: Integer;
begin
  ListBox1.BeginUpdate;
  ListBox1.Clear;
  for I := 0 to TMSFMXNativeiCloudDocument1.DocumentCount - 1 do
  begin
    doc := TMSFMXNativeiCloudDocument1.DocumentByIndex[I];
    // ListBox1.Items.Add(doc.DisplayName);
    ListBox1.Items.Add(doc.FileSystemName);
  end;
```

```

ListBox1.EndUpdate;
end;

```

In the code sample we specify the `FileSystemName` which includes the extension, but you can also use the `DisplayName` which is a more meaningful name given to a document without the need for an extension.

## 2.45.6 Notes sample

To have a better understanding how the initialization process works, how to add new, delete or update existing documents, have a look at the iCloud Documents demo, which demonstrates the management of automatically synchronized notes throughout various devices which are all connected to the the same iCloud document storage container.

The demo also specifies an entitlements file that is used to sign the application to allow iCloud access. Below is more information on how to create provisioning profiles and correctly sign your application.

## 2.45.7 Entitlements

Before iCloud can be used in your application you need to enable it and sign your application. Additional information about enabling iCloud and incorporating it into your application can be found on the following page:

<https://developer.apple.com/library/ios/documentation/General/Conceptual/iCloudDesignGuide/Chapters/iCloudFundamentals.html>

After reading the guide, you will need to perform 2 steps: signing your device, and adding an entitlements file that adds the necessary keys to gain access to the iCloud storage. As a helper sample we have included an iCloud Document demo project that demonstrates how the Entitlements file is added.

When opening the Entitlements file (iCloud.entitlements) you will notice placeholders that need to be filled in with a combination of the Team-ID and the Bundle Identifier

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>get-task-allow</key>
  <true/>
  <key>com.apple.developer.ubiquity-container-identifiers</key>
  <array>
    <string>$(TeamIdentifierPrefix)com.mycompany.myapplication</string>
  </array>
  <key>com.apple.developer.ubiquity-kvstore-identifier</key>
  <string>$(TeamIdentifierPrefix)com.mycompany.myapplication</string>
</dict>
</plist>

```

The first key is to allow the debugger to access the application. This is inherited from the default entitlements file that is distributed when deploying your application. The `com.apple.developer.ubiquity-container-identifiers` and the `com.apple.developer.ubiquity-kvstore-identifier` keys are used to access iCloud. Here you need to specify the correct Team Identifier Prefix and the Bundle Identifier that matches your Application ID, used in the generation of the provisioning profile. Below is a sample of the Application ID at [developer.apple.com](https://developer.apple.com), used to generate a provisioning profile to sign your application.

FireMonkeySample
com.tmssoftware.FireMonkeySample

ID

Name: FireMonkeySample

Prefix: ABC123

ID: com.tmssoftware.FireMonkeySample

---

**Application Services:**

Service	Development	Distribution
<b>Data Protection</b>	<input type="radio"/> Disabled	<input type="radio"/> Disabled
<b>Game Center</b>	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
<b>iCloud</b>	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
<b>In-App Purchase</b>	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
<b>Inter-App Audio</b>	<input type="radio"/> Disabled	<input type="radio"/> Disabled
<b>Passbook</b>	<input type="radio"/> Disabled	<input type="radio"/> Disabled
<b>Push Notifications</b>	<input checked="" type="radio"/> Configurable	<input checked="" type="radio"/> Configurable

If the prefix is `ABC123` and the ID is `com.tmssoftware.FireMonkeySample`. The correct Entitlements.plist file would have `ABC123.com.tmssoftware.FireMonkeySample` as substitute for `$(TeamIdentifierPrefix)com.mycompany.myapplication`.

## 2.46 TMSFMXNativePDFLib

---

### 2.46.1 Usage

---

The `TMSFMXNativePDFLib` component is used to create rich pdf documents with support for text flow in multiple columns, rich text, images and various shapes with fill stroke and gradient colors.

## 2.46.2 Methods

Methods name	Description
BeginDocument(FileName: String = "");	Creates a new PDF document. If the FileName parameter is not specified, the PDF document is created in memory and returns a <code>TMemoryStream</code> instance when calling <code>EndDocument</code> .
CloseDocument	Close an active PDF document.
DrawPage(PageIndex: Integer)	Draws an existing PDF page to a new PDF document.
EndDocument	Ends the document and writes all remaining data from memory to a file or a memstream, depending on the chosen action in <code>BeginDocument</code> .
GetDocumentInfo	When an existing PDF document is opened, this method retrieves the document information such as the Author, Creator, Title, Subject ... When calling this method, the existing data is overwritten and applied when creating a new document.
GetPageCount	When an existing PDF document is opened this method returns the number of pages.
GetPageInfo(PageIndex: Integer)	When an existing PDF document is opened, the <code>GetPageInfo</code> method returns the various boxes ( <code>MediaBox</code> , <code>CropBox</code> , <code>TrimBox</code> , <code>ArtBox</code> and <code>BleedBox</code> ) that are used in that page. These boxes can be used to determine the page size and orientation. When calling this method the previous data is overwritten and applied when creating a new page.
IsDocumentOpened	<code>Boolean</code> to determine if a PDF document was already opened with <code>OpenDocument</code> .
NewPage	Creates a new PDF page.
OpenDocument(FileName: String)	Opens an existing PDF document from a file.
OpenDocument(FileStream: TMemoryStream)	Opens an existing PDF document from a <code>TMemoryStream</code> .
SaveDocumentFromStream(FileStream: TMemoryStream; FileName: String);	Saves an existing PDF document from a <code>TMemoryStream</code> to a file.
UnlockWithPassword(Password: String): Boolean;	When an existing PDF document is opened, the contents might be encrypted when calling <code>GetDocumentInfo</code> . This method unlocks the document with a password and returns a <code>Boolean</code> if the document is unlocked successfully.

## 2.46.3 Public Properties

Properties name	Description
MediaBox, TrimBox, BleedBox, ArtBox, CropBox	<p data-bbox="740 338 1449 405">Various boxes that can be used to retrieve / set information from / to a page or document that is created / opened.</p> <p data-bbox="740 456 1449 600">The <code>MediaBox</code> is used to specify the width and height of the page. The <code>MediaBox</code> is the largest page box in a PDF. The other page boxes can equal the size of the <code>MediaBox</code> but they cannot be larger.</p> <p data-bbox="740 651 1449 719">The <code>CropBox</code> defines the region to which the page contents are to be clipped.</p> <p data-bbox="740 770 1449 837">The <code>BleedBox</code> determines the region to which the page contents needs to be clipped when output in a production environment.</p> <p data-bbox="740 889 1449 992">The <code>TrimBox</code> defines the intended dimensions of the finished page. Contrary to the <code>CropBox</code>, the <code>TrimBox</code> is very important because it defines the actual page size.</p> <p data-bbox="740 1043 1449 1111">The <code>ArtBox</code> is a bit of a special case. It can define a region within a page that is of special interest.</p>
ModificationDate	The date the PDF document was modified.
CreationDate	The date the PDF document was created.
Producer	The producer of the PDF document.

## 2.46.4 Properties

---



Properties name	Description
AllowsCopying	Enable or disable copying on a new PDF password protected document.
AllowsPrinting	Enable or disable printing on a new PDF password protected document.
Author	The author of the PDF document.
Creator	The creator of the PDF document.
FillColor	The fill color used for drawing shapes and drawing text. This property is also the start color for a gradient.
FillColorTo	The end color for a gradient.
Font	The font used when drawing text in a document.
Footer	The footer drawn at the bottom of each new page.
FooterAlignment	The alignment of the footer text.
FooterMargins	The margins applied to the footer rectangle.
FooterSize	The height of the footer rectangle.
Header	The header drawn at the bottom of each new page.
HeaderAlignment	The alignment of the header text.
HeaderMargins	The margins applied to the header rectangle.
HeaderSize	The height of the header rectangle.
Keywords	The keywords of the PDF document.
LineBreakMode	The linebreakmode when drawing text in a PDF document.
LineWidth	The width of the stroke when drawing shapes or the width of the line when drawing lines.
Orientation	The orientation of a page / document. This property cannot be used to retrieve the orientation of a page, only to modify the box rectangles that are used when creating a new page. Read out the box rectangle properties after opening a document and calling <a href="#">GetPageInfo</a> , to get more information about the page size and orientation.
OwnerPassword	The owner password of the PDF document. You can set an owner password to keep other people from printing, copying or modifying text, adding or deleting pages in your PDF files.
PageSize	The page size of a page / document. This property cannot be used to retrieve the page size of a page, only to modify the box rectangles that are used when creating a new page. Read out the box rectangle properties after opening a document and calling <a href="#">GetPageInfo</a> , to get more information about the page size and orientation.
StrokeColor	The color of the stroke when drawing a shape or line.
Subject	The subject of the PDF document.

Properties name	Description
Title	The title of the PDF document.
UserPassword	The user password of the PDF document. This kind of password is used to help prevent opening or viewing your PDF. You can unlock your pdf passing this password in the as a parameter of the <a href="#">UnlockWithPassword</a> method.

## 2.46.5 Creating a new document

The code snippets below demonstrates how to create a new document based on a file or a memory stream. If the file exist the PDF document contents are cleared.

```
TMSFMXNativePDFLib1.BeginDocument('FileName');
TMSFMXNativePDFLib1.NewPage;
TMSFMXNativePDFLib1.EndDocument;
```

To create a new document in memory use the following code:

```
var
  ms: TMemoryStream;
begin
  TMSFMXNativePDFLib1.BeginDocument;
  TMSFMXNativePDFLib1.NewPage;
  ms := TMSFMXNativePDFLib1.EndDocument;
end;
```

## 2.46.6 Opening an existing document

The code snippet below demonstrates how to open an existing document based on a file or a memory stream.

```
TMSFMXNativePDFLib1.OpenDocument('FileName');
if TMSFMXNativePDFLib1.UnlockWithPassword('Password') then //optional password unlocking
begin
  TMSFMXNativePDFLib1.GetDocumentInfo; // get document information //such as the Author, Title, ...
  TMSFMXNativePDFLib1.GetPageInfo(1); // get page informaton such as //the MediaBox, CropBox, ...
end;
TMSFMXNativePDFLib1.CloseDocument;
```

Opening a document from a memory stream is based on the same code but with a different [OpenDocument](#) overload.

## 2.46.7 Drawing pages from an existing PDF document

Editing a PDF page or document is only possible if the page is drawn on a different context in a new PDF Document. The reason for editing might be to add watermarks, to merge multiple documents, add or remove pages. The sample below copies the PDF pages from an existing document to a new document.

```
var
  I: Integer;
begin
  TMSFMXNativeMacPDFLib1.OpenDocument('Existing.pdf');
```

```

TMSFMXNativeMacPDFLib1.BeginDocument('New.pdf');
for I := 1 to TMSFMXNativeMacPDFLib1.GetPageCount do
begin
  //copy page information
  TMSFMXNativeMacPDFLib1.GetPageInfo(I);
  //add page to new document
  TMSFMXNativeMacPDFLib1.NewPage;
  //draw page from existing document
  TMSFMXNativeMacPDFLib1.DrawPage(I);
  //additional manipulation / drawing
  //...
end;
TMSFMXNativeMacPDFLib1.EndDocument;
TMSFMXNativeMacPDFLib1.CloseDocument;
end;

```

## 2.46.8 Graphics Library

The above table does not list all methods that are available in the PDF rendering library. The PDF rendering library inherits from the Graphics Library and is able to draw images, shapes / lines with solid / gradient colors and plain text. All methods start with Draw and can be used within a new PDF page. The Graphics Library also supports more complex shapes drawn within a path. The code below demonstrates how this can be achieved.

```

TMSFMXNativePDFLib1.BeginDocument('FileName');
TMSFMXNativePDFLib1.NewPage;
TMSFMXNativePDFLib1.FillColor := TAlphaColorRec.Red;
TMSFMXNativePDFLib1.StrokeColor := TAlphaColorRec.Darkred;
TMSFMXNativePDFLib1.LineWidth := 3;
TMSFMXNativePDFLib1.DrawPathBegin;
TMSFMXNativePDFLib1.DrawPathMoveToPoint(PointF(200, 200));
TMSFMXNativePDFLib1.DrawPathAddCurveToPoint(PointF(250, 150), PointF(325, 250), PointF(200, 300));
TMSFMXNativePDFLib1.DrawPathAddCurveToPoint(PointF(75, 250), PointF(150, 150), PointF(200, 200));
TMSFMXNativePDFLib1.DrawPathClose;
TMSFMXNativePDFLib1.DrawPathEnd;
TMSFMXNativePDFLib1.EndDocument;

```

## 2.46.9 Graphics Library Rich Text

The Graphics Library also supports rendering rich text. For more information, please read the [TTMSFMXNativeUIRichTextView](#) chapter that explains the capabilities of rendering rich text. The method name that is being used to render rich text is "DrawRichText". All properties related to rich text can be accessed at the [RichText](#) function directly available from the PDF Library component. Below is a sample that demonstrates this.

```

TMSFMXNativePDFLib1.BeginDocument('FileName');
TMSFMXNativePDFLib1.NewPage;
TMSFMXNativePDFLib1.RichText.Text := 'Hello World';
TMSFMXNativePDFLib1.RichText.SetBold;
TMSFMXNativePDFLib1.RichText.SetForegroundColor(TAlphaColorRec.Red, 0, 5);
TMSFMXNativePDFLib1.DrawRichText(RectF(50, 50, 150, 100));
TMSFMXNativePDFLib1.EndDocument;

```

## 2.46.10 Text Flow

Starting from iOS 7 the PDF Rendering Library supports drawing text in multiple columns. The code below demonstrates how easy it is to specify text, a rectangle and the amount of columns. The text flow feature is also available for rich text.

```

lorem := 'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has
been the industry's standard dummy text ever since '+
'the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen
book. It has survived not only five centuries, but also '+
'the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the
1960s with the release of Letraset sheets containing Lorem '+
'Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including
versions of Lorem Ipsum. It is a long established fact that'+

'a reader will be distracted by the readable content of a page when looking at its layout. The point
of using Lorem Ipsum is that it has a more-or-less normal'+
'distribution of letters, as opposed to using ''Content here, content here'', making it look like
readable English. '+
'Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model
text, and a search for ''lorem ipsum'' '+
'will uncover many web sites still in their infancy. Various versions have evolved over the years,
sometimes by accident, sometimes on purpose (injected humour and the like).';

r := TMSFMXNativePDFLibl.MediaBox;
InflateRect(r, -50, -50);
TMSFMXNativePDFLibl.BeginDocument('FileName');
TMSFMXNativePDFLibl.NewPage;
TMSFMXNativePDFLibl.DrawText(lorem, RectF(r.Left, R.Top, r.Right, R.Top + 250), 3);
TMSFMXNativePDFLibl.EndDocument;

```

### 2.46.11 Text Calculation And Overflow

Each `DrawText / DrawRichText` call has a number of overloads to draw at a point, in a rectangle or with text flow. Additional default parameters `Calculate` and `DetectOverflow` can be used to calculate the size of the text and the detect the number of characters that remain when drawing the text inside a rectangle with overflow capabilities. Specifying a True value to these parameters forces the method to calculate instead of drawing the text.

### 2.46.12 Images

The PDF Rendering Library supports drawing images at a specific point, with aspect ratio in a rectangle and optional PNG and JPG quality. Specifying JPG as drawing type has an additional `Quality` parameter from `0` to `1` where `0` is the lowest quality when drawing.

## 2.47 TMSFMXNativeMultipeerConnectivity

---

### 2.47.1 Usage

The `TMSFMXNativeMultipeerConnectivity` component

### 2.47.2 Methods

Methods name	Description
SearchForPeers	Searches and displays available peers configured with the same <code>ServiceType</code> property.
SendResource(AFile: String; APeer: MCPeerID)	Sends a resource file to a specific peer.
SendResourceToAllPeers(AFile: String)	Sends a resource file to all peers.
SendString(AValue: String; APeer: MCPeerID)	Sends a <code>String</code> to a specific peer.
SendStringToAllPeers(AValue: String); overload	Sends a <code>String</code> to all peers.
SendInteger(AValue: Integer; APeer: MCPeerID)	Sends an <code>Integer</code> to a specific peer.
SendIntegerToAllPeers(AValue: Integer)	Sends an <code>Integer</code> to all peers.
SendBoolean(AValue: Boolean; APeer: MCPeerID)	Sends a <code>Boolean</code> to a specific peer.
SendBooleanToAllPeers(AValue: Boolean)	Sends a <code>Boolean</code> to all peers.
SendDouble(AValue: Double; APeer: MCPeerID)	Sends a <code>Double</code> to a specific peer.
SendDoubleToAllPeers(AValue: Double)	Sends a <code>Double</code> to all peers.
SendObject(AValue: TMemoryStream; APeer: MCPeerID)	Sends a memory stream object to a specific peer.
SendObjectToAllPeers(AValue: TMemoryStream)	Sends a memory stream object to all peers.
PeerCount: Integer	The number of connected peers.

## 2.47.3 Public Properties

Properties name	Description
AdvertiserAssistant	Assistant that handles users' responses and presents incoming peer connections through the <code>BrowserViewController</code> .
BrowserViewController	The controller that is used to display a list of available peers, limited to the <code>MinimumNumberOfPeers</code> and the <code>MaximumNumberOfPeers</code> properties. Already connected peers are also displayed in this window. The <code>BrowserViewController</code> is shown when calling <code>SearchForPeers</code> .
Peers[Index: Integer]	
PeerDisplayNames[Index: Integer]	Returns the name of the connected peer at a specific index.
PeerID	Your own created peer ID used to connect to other peers. The display name of the peer can be set at design time or runtime with the <code>MyPeerID</code> property when the <code>MyPeerIDKind</code> property is set to <code>pidkCustom</code> . By default the peer is configured to use the name of the device.
Session	The current session, with a session service type. The session name can be set with the property <code>ServiceType</code> .

## 2.47.4 Properties

Properties name	Description
MaximumNumberOfPeers	The maximum number of allowed peers in a session, which is <code>8</code> by default.
MinimumNumberOfPeers	The minimum number of required peers in a session, which is <code>2</code> by default. This number already incorporates your own peer connection.
MyPeerID	The name of your own peer ID, used to display to other peers when establishing a connection. The display name of the peer is set to the device name by default but can be changed through this property after setting the <code>MyPeerIDKind</code> property to <code>pidkCustom</code> .
MyPeerIDKind	The kind of displayname your own peer will have when establishing a connection with other peers. More information can be found at the <code>MyPeerID</code> property explanation.
SendDataMode	The mode used to send data, such as strings, integers, booleans and memory streams reliable or unreliable. The reliable method is slower than the unreliable method but additionally verifies if the sent data is received correctly.
ServiceType	The type of service the peer is offering when establishing a connection to other peers. This property is used to create a session. Only peers with the same <code>ServiceType</code> property can connect to each other.

## 2.47.5 Events

---

Events name	Description
OnBrowserViewControllerDidFinish	Event called when the <code>BrowserViewController</code> finished searching and connecting peers. When the <code>MaximumNumberOfPeers</code> has been reached, the <code>BrowserViewController</code> will automatically dismiss and call this event. This event is also called when clicking the Done button.
OnBrowserViewControllerWasCancelled	Event called when the cancel button of the <code>BrowserViewController</code> has been clicked.
OnDidChangeState	Event called when the state of one of the connected peers has changed. The state of the peer can be disconnected, connecting or connected. This event can be called multiple times with different peer and state parameters.
OnDidReceiveBoolean	Event called when a <code>Boolean</code> is received from a specific peer.
OnDidReceiveDouble	Event called when a <code>Double</code> is received from a specific peer.
OnDidReceiveInteger	Event called when an <code>Integer</code> is received from a specific peer.
OnDidReceiveObject	Event called when a memorystream object is received from a specific peer.
OnDidReceiveResource	Event called when a resource file is completely received. Through this event, the automatically created temporary file will be saved in the Documents folder of the application. This behavior can be changed by changing the value of the <code>AllowSave</code> parameter. The filename that is being used to save the temporary file in the documents folder can be overridden by changing the <code>ASaveFileName</code> parameter.
OnDidReceiveString	Event called when a <code>String</code> is received from a specific peer.
OnDidSendResource	Event called when a resource file is sent and has successfully reached the peer it was sent to.
OnDidStartReceivingResource	Event called when a resource will be received from a specific peer.
OnError	Event called when an error occurred during sending or receiving data and resource files.
OnReceiveResource	Event called multiple times with the progress of the resource file that is being received, sent by a specific peer. Through this event, the <code>AProgress</code> parameter can be used to indicate the receiving progress. The <code>ACancel Boolean</code> parameter can be used to cancel receiving a resource file.
OnSendResource	Event called multiple times with the progress of the resource file that is being sent to a specific peer. Through this event, the <code>AProgress</code> parameter can be used to indicate the sending progress. The <code>ACancel Boolean</code> parameter can be used to cancel sending a resource file.



## 2.47.6 Managing peers

Before data and/or resource files can be sent to single peer or multiple peers, the peer(s) must first be connected to a session based on the the `ServiceType` property. The `ServiceType` property is preset with “tms-peertopeer” and can be modified at designtime. This property identifies your session as an entry point for other peers. A session with a different `ServiceType` property, will not be able to identify the peers managed by the session that is created with “tms-peertopeer”.

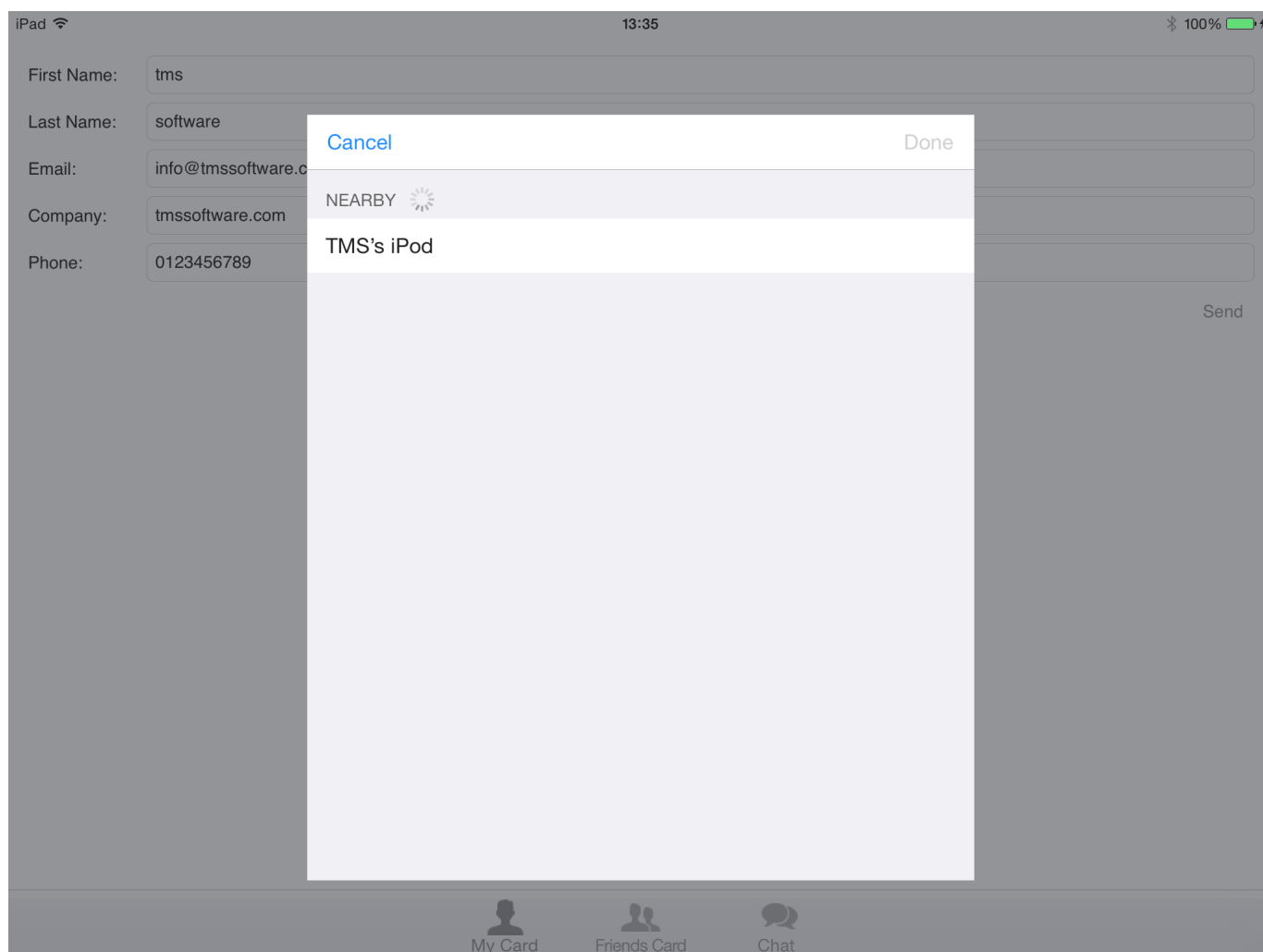
Each session has it's own peer id, to identify itself to other peers.

By default, the `PeerID` instance is assigned a displayname. The displayname is set to the device name by default, but can be changed to a custom value, by setting the `MyPeerIDKind` to `pidkCustom` and setting the `MyPeerID` property to a value of choice.

After properly determining the `ServiceType` and the `MyPeerID` properties, the application is ready to create the session and browse for other peers. Peers can be searched by calling the following code:

```
TMSFMXNativeMultipeerConnectivity1.SearchForPeers
```

`SearchForPeers` will automatically popup the `BrowserViewController` instance, which will handle the connection of all peers within a session. Tap on the peers that are available for a connection, and the `BrowserViewController` will handle and maintain the connection.



When a peer is connected, the application is ready to send data or resource files to one or multiple peers. To know the connected peers, you can use the `PeerCount` function and the `Peers` property to retrieve a connected peer by specifying an index. To find out the names of the connected peers and display them in a list, you can use the `PeerDisplayNames` property with the same approach.

The `PeerCount` function, `Peers` and `PeerDisplayNames` properties will automatically update as the connections are automatically managed by the `TMSFMXNativeMultipeerConnectivity` component. To know the state of one or multiple peers, you can use the `OnDidChangeState` event, that will allow you to monitor the state of each peer, whether it is disconnected, connecting or connected.

## 2.47.7 Sending Data

To send data, you can use one of the multiple methods specified and explained in the methods table. Below is a sample to send a String to all connected peers.

```
TMSFMXNativeMultipeerConnectivity1.SendStringToAllPeers('Hello World');
```

If you wish to send a `Boolean` to a specific peer you can use the code below:

```
var
  peer: MCPeerID;
begin
  peer := TMSFMXNativeMultipeerConnectivity1.Peers[0];
  TMSFMXNativeMultipeerConnectivity1.SendBoolean(true, peer);
end;
```

## 2.47.8 Receiving Data

Receiving data is done through one of the various events. Each `Send*` method has a equivalent for receiving that specific type of data. Sending a string can be received with the `OnDidReceiveString`, while sending a `Boolean` can be received by implementing the `OnDidReceiveBoolean`. Below is a sample that displays the String value with the `Peer` displayname as the text of a label.

```
procedure TForm1.TMSFMXNativeMultipeerConnectivity1DidReceiveString(
  Sender: TObject; AValue: string;
  APeer: TTMSFMXNativeMultipeerConnectivityPeer);
begin
  Label1.Text := 'Received ' + AValue + ' from ' + APeer.DisplayName;
end;
```

## 2.47.9 Sending and Receiving Files

Sending and Receiving files is done with the same approach as sending and receiving data, with the possibility to cancel and monitor progress of a send and/or receive operation. Various events are published to manage this operation. More information can be found at the [Method & Events](#) table.

## 2.48 TMSFMXNativeCLLocationManager

---

### 2.48.1 Usage

---

The `TMSFMXNativeCLLocationManager` component is the central point for configuring the delivery of location- and heading-related events to your app. You use this component to establish the parameters that determine when location and heading events should be delivered and to start and stop the actual delivery of those events.

## 2.48.2 Methods

Methods name	Description
AuthorizationStatus	The status of the location manager. When your application isn't authorized to receive location you will receive an <code>asAuthorizationStatusRestricted</code> / <code>asAuthorizationStatusDenied</code> value. When the value is <code>asAuthorizationStatusNotDetermined</code> your application needs to ask permissions before starting to monitor location and/or heading updates.
DismissHeadingCalibrationDisplay	When monitoring heading updates the location manager might display a heading calibration window which can be dismissed with this method. The calibration window is only shown when True is returned in the <code>OnShouldDisplayHeadingCalibration</code> event.
Heading	The last active heading managed by the location manager.
HeadingAvailable	Verify if the location manager can monitor heading updates.
Location	The last active location managed by the location manager.
LocationManager	Returns a reference to the native <code>CLLocationManager</code> instance.
LocationServicesEnabled	Verify if the location services are enabled before starting to monitor for location updates.
RequestAlwaysAuthorization	When the <code>AuthorizationStatus</code> is not determined, request an "Always" authorization status for your application with this method. The <code>OnDidChangeAuthorizationStatus</code> is called when the status changes.
RequestWhenInUseAuthorization	When the <code>AuthorizationStatus</code> is not determined, request a "When In Use" authorization status for your application with this method. The <code>OnDidChangeAuthorizationStatus</code> is called when the status changes.
SignificantLocationChangeMonitoringAvailable	Verify if the location manager can monitor location updates, which will update after a significant difference is detected between the initial value and the value that is monitored.
StartMonitoringSignificantLocationChanges	Start monitoring for location changes that are only retrieved when a significant difference is detected.
StartUpdatingHeading	Start monitoring for heading updates.
StartUpdatingLocation	Start monitoring for location changes.
StopMonitoringSignificantLocationChanges	Stop monitoring for significant location changes.
StopUpdatingHeading	Stop monitoring for heading updates.
StopUpdatingLocation	Stop monitoring for location changes.

## 2.48.3 Properties

Properties name	Description
ActivityType	The type of activity that is executed for monitoring location and heading updates through the location manager.
DesiredAccuracy	The accuracy of the location data in meters. If the value is <code>-1</code> , the location manager automatically determines the best accuracy for your device.
DistanceFilter	The minimum distance (measured in meters) a device must move horizontally before an update event is generated.
HeadingOrientation	The device orientation to use when computing heading values.
PausesLocationUpdatesAutomatically	Property to configure automatic pausing and resuming of location changes.

## 2.48.4 Events

Events name	Description
OnDidChangeAuthorizationStatus	Event called when the authorization status for your app changes.
OnDidFailWithError	Event called when the location manager fails updating location/heading.
OnDidPauseLocationUpdates	Event called automatically when the location manager has paused location update changes.
OnDidResumeLocationUpdates	Event called automatically when the location manager has resumed location update changes.
OnDidUpdateHeading	Event called when the heading changes.
OnDidUpdateLocations	Event called when the location changes.
OnShouldDisplayHeadingCalibration	Event that could be called if the device needs to calibrate the heading when monitored by your application.

## 2.48.5 Sample authorization and managing the location updates

The code below verifies if the location services are enabled and if your application is authorized to use location updates. The first time the application starts, the user will be prompted with an authorization dialog which asks permissions to use location updates. Afterwards, the method `StartUpdatingLocation` will be called and the annotation with the location of the device will be added to the map.

```
if TMSFMXNativeCLLocationManager1.LocationServicesEnabled then
begin
  if TMSFMXNativeCLLocationManager1.AuthorizationStatus = asAuthorizationStatusNotDetermined then
    TMSFMXNativeCLLocationManager1.RequestAlwaysAuthorization
  else
    StartLocationUpdates;
end;
```

```

procedure TForm1.StartLocationUpdates;
begin
    TMSFMXNativeCLLocationManager1.StartUpdatingLocation;
end;
procedure TForm1.TMSFMXNativeCLLocationManager1DidChangeAuthorizationStatus (
    Sender: TObject;
    AAuthorizationStatus: TTMSFMXNativeCLLocationManagerAuthorizationStatus);
begin
    if AAuthorizationStatus = asAuthorizationStatusAuthorizedAlways then
        StartLocationUpdates;
end;

procedure TForm1.TMSFMXNativeCLLocationManager1DidUpdateLocations (
    Sender: TObject;
    ALocations: TArray<FMX.TMSNativeUICore.TTMSFMXNativeCLLocation>);
var
    ann: TTMSFMXNativeMKAnnotation;
begin
    if Length(ALocations) > 0 then
    begin
        TMSFMXNativeMKMapView1.BeginUpdate;
        if TMSFMXNativeMKMapView1.Annotations.Count = 0 then
            ann := TMSFMXNativeMKMapView1.Annotations.Add
        else
            ann := TMSFMXNativeMKMapView1.Annotations[0];

        ann.Location := MakeMapLocation(ALocations[0].Coordinate.Latitude,
ALocations[0].Coordinate.Longitude);
        TMSFMXNativeMKMapView1.SetCenterLocation(ann.Location, True);
        TMSFMXNativeMKMapView1.EndUpdate;
    end;
end;

```

## 2.49 TMSFMXNativeCMMotionManager

---

### 2.49.1 Usage

---

The `TMSFMXNativeCMMotionManager` component is the gateway to the motion services provided by iOS. These services provide an app with accelerometer data, rotation-rate data, magnetometer data, and other device-motion data such as attitude. These types of data originate with a device's accelerometers and (on some models) its magnetometer and gyroscope.

## 2.49.2 Methods

Methods name	Description
AccelerometerActive	Verify if the accelerometer is active. (Has an active reading between starting and stopping).
AccelerometerAvailable	Verify if the accelerometer is available.
AccelerometerData	Returns the most recent accelerometer data.
DeviceMotion	Returns the most recent device motion data.
DeviceMotionActive	Verify if the device motion is active. (Has an active reading between starting and stopping).
DeviceMotionAvailable	Verify if the device motion is active. (Has an active reading between starting and stopping).
GyroActive	Verify if the gyroscope is active. (Has an active reading between starting and stopping).
GyroAvailable	Verify if the gyroscope is available.
GyroData	Returns the most recent gyroscope data.
MagnetometerActive	Verify if the magnetometer is active. (Has an active reading between starting and stopping).
MagnetometerAvailable	Verify if the magnetometer is available.
MagnetometerData	Returns the most recent magnetometer data.
MotionManager	A reference to the native <code>CMMotionManager</code> class.
StartAccelerometerUpdates	Starts monitoring accelerometer data changes.
StartDeviceMotionUpdates	Starts monitoring device motion changes.
StartGyroUpdates	Starts monitoring gyroscope data changes.
StartMagnetometerUpdates	Starts monitoring for magnetometer data changes.
StopAccelerometerUpdates	Stops monitoring accelerometer data changes.
StopDeviceMotionUpdates	Stops monitoring device motion changes.
StopGyroUpdates	Stops monitoring gyroscope data changes.
StopMagnetometerUpdates	Stops monitoring for magnetometer data changes.



## 2.49.3 Properties

Properties name	Description
AccelerometerUpdateInterval	The interval in seconds for providing updates from the accelerometer.
DeviceMotionUpdateInterval	The interval in seconds for providing updates from the device motion.
GyroUpdateInterval	The interval in seconds for providing updates from the gyroscope.
MagnetometerUpdateInterval	The interval in seconds for providing updates from the magnetometer.

## 2.49.4 Events

Events name	Description
OnAccelerometerError	Event called when an error occurred during accelerometer data changes.
OnDeviceMotionError	Event called when an error occurred during device motion changes.
OnGetAccelerometerData	Event called when data from the accelerometer changes.
OnGetDeviceMotion	Event called when device motion changes.
OnGetGyroData	Event called when data from the gyroscope changes.
OnGetMagnetometerData	Event called when data from the magnetometer changes.
OnGyroError	Event called when an error occurred during gyroscope data changes.
OnMagnetometerError	Event called when an error occurred during magnetometer data changes.

## 2.49.5 Sample with Device Motion

The code below verifies if Device Motion is available and then executes the `StartDeviceMotionUpdates` method which then handles the data through an anonymous method which positions the Ellipse base on the `TTMSFMXNativeCMDeviceMotion` data record. An alternative would be to move the code that updates the Ellipse inside the `OnGetDeviceMotion` event.

```

if TMSFMXNativeCMMotionManager1.DeviceMotionAvailable then
begin
  TMSFMXNativeCMMotionManager1.StartDeviceMotionUpdates(
    procedure (AData: TTMSFMXNativeCMDeviceMotion)
    begin
      Ellipse1.Position.X := Max(0,Min(Panel1.Width - Ellipse1.Width, Ellipse1.Position.X +
(AData.Attitude.Roll * 20)));
      Ellipse1.Position.Y := Max(0,Min(Panel1.Height - Ellipse1.Height, Ellipse1.Position.Y +
(AData.Attitude.Pitch * 20)));
      if PtInRect(RectF(Rectangle1.Position.X, Rectangle1.Position.Y, Rectangle1.Position.X +
Rectangle1.Width,
      Rectangle1.Position.Y + Rectangle1.Height), PointF(Ellipse1.Position.X, Ellipse1.Position.Y))
    then
      Label2.Text := inttostr(strtoint(Label2.Text) + 1)
    else
      Label2.Text := '0';
    end
  )
end

```

```
);  
end  
else  
    ShowMessage('Device Motion is not available on this device');
```

## 2.50 TMSFMXNativeCMAltimeter

### 2.50.1 Usage

Use the `TMSFMXNativeCMAltimeter` component to detect changes of altitude-related data to your app. (iOS 8 or later)

### 2.50.2 Methods

Methods name	Description
Altimeter	Returns a reference to native the <code>CMAltimeter</code> class.
RelativeAltitudeAvailable	Verify whether your device is capable of monitoring changes in relative altitude.
StartRelativeAltitudeUpdates	Starts monitoring changes in relative altitude.
StopRelativeAltitudeUpdates	Stops monitoring changes in relative altitude.

### 2.50.3 Events

Events name	Description
OnRelativeAltitudeChanged	Event called when the relative altitude changes.
OnRelativeAltitudeError	Event called when an error occurred during relative altitude data changes.

### 2.50.4 Sample obtaining relative altitude updates

Through an anonymous method

```
if TMSFMXNativeCMAltimeter1.RelativeAltitudeAvailable then
begin
  TMSFMXNativeCMAltimeter1.StartRelativeAltitudeUpdates(
    procedure (AData: TTMSFMXNativeCMAltitudeData)
    begin
      Label1.Text := 'Relative altitude is ' + floattostr(AData.RelativeAltitude);
      Label2.Text := 'Pressure is ' + floattostr(AData.Pressure);
    end
  )
end;
```

Through an event

```
if TMSFMXNativeCMAltimeter1.RelativeAltitudeAvailable then
  TMSFMXNativeCMAltimeter1.StartRelativeAltitudeUpdates;

procedure TForm107.TMSFMXNativeCMAltimeter1RelativeAltitudeChanged(
  Sender: TObject; AData: TTMSFMXNativeCMAltitudeData);
begin
  Label1.Text := 'Relative altitude is ' + floattostr(AData.RelativeAltitude);
```

```
Label2.Text := 'Pressure is ' + floattostr(AData.Pressure);  
end;
```

## 2.51 TMSFMXNativeLocalAuthentication

---

### 2.51.1 Usage

The Local Authentication framework provides facilities for requesting authentication from users with specified security policies. The Local Authentication framework can be used to authenticate the user via Touch ID. (iOS 8 or later)

### 2.51.2 Methods

Methods name	Description
Authenticate	Displays a dialog to allow authentication via Touch ID.

### 2.51.3 Events

Events name	Description
OnAuthenticateError	Event called when the user cancelled the dialog, pressed an alternative option such as “Enter password” or another error occurred.
OnAuthenticateSuccess	Event called when successfully authenticated via Touch ID.

## 2.52 TMSFMXNativeUIDocumentInteractionController

---

### 2.52.1 Usage

A document interaction controller provides in-app support for managing user interactions with files in the local system. For example, an email program might use this class to allow the user to preview attachments and open them in other apps. Use this component to present an appropriate user interface for previewing, opening, copying, or printing a specified file.

### 2.52.2 Methods

The `TMSFMXNativeUIDocumentInteractionController` consists of a series of show and hide methods to display either an option or a preview menu. Some show methods can be used in combination with other controls such as another `TMSFMXNativeUIBaseControl` descendant or a `UIBarButtonItem` from a `ToolBar`.

### 2.52.3 Properties

Properties name	Description
File	The file to display in the open, options, or preview menu.
FileUTI	The UTI for the file to specify additional information. (automatically extracted from the file by default)
FileName	The name for the file. (automatically extracted from the file by default)

## 2.53 TMSFMXNativeAVPlayerViewController

---

### 2.53.1 Usage

A `TMSNativeAVPlayerViewController` displays the video content of an `AVPlayer` object along with system-supplied playback controls. When you use a player view controller, the system makes its media content available for the user to play on the screen of the playback device or on a second screen such as Apple TV. Starting in iOS 9, supported iPad models also provide Picture in Picture playback from a player view controller.

### 2.53.2 Methods

Methods name	Description
Hide	When the <code>ShowInView</code> property is set to <code>False</code> , the <code>PlayerViewController</code> can be presented fullscreen with the <code>Show</code> method. The <code>Hide</code> method will hide the presented <code>PlayerViewController</code> .
InitializePlayback	Initializes audio, airplay and picture in picture (iOS 9) support.
Pause	Pauses the audio or video.
Play	Plays the audio or video.
Player	Returns a native reference to the <code>AVPlayer</code> , automatically created when using an <code>AVPlayerViewController</code> .
PlayerViewController	Returns a native reference to the <code>AVPlayerViewController</code> that holds a reference to the <code>AVPlayer</code> instance.
ReadyForDisplay: Boolean;	Returns a <code>Boolean</code> whether the audio/video is ready to be displayed.
Show	When the <code>ShowInView</code> property is set to <code>False</code> , the <code>PlayerViewController</code> can be presented fullscreen with this method.
Stop	Stops playing the audio or video.
VideoBounds: CGRect;	Returns the current video rectangle used inside the <code>PlayerViewController</code> .

## 2.53.3 Properties

Properties name	Description
AllowsPictureInPicturePlayback	Sets a <code>Boolean</code> whether picture in picture is supported (iOS 9).
Location	The location of the local video / audio file.
ShowInView	Shows the player in the view controller (default). When set to False, the player can be shown full screen when using the <code>Show</code> method.
ShowPlaybackControls	Sets a <code>Boolean</code> whether the playback controls are visible or not.
URL	The URL of the remote video / audio file.
VideoGravity	The aspect ratio of the video inside the view controller.

## 2.53.4 Events

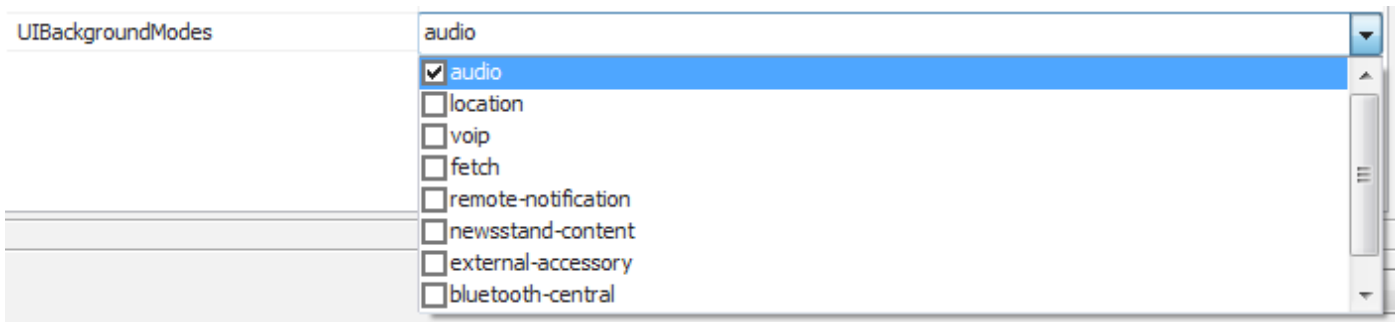
Events name	Description
OnDidStartPictureInPicture	Event called when picture in picture is started.
OnDidStopPictureInPicture	Event called when picture in picture is stopped.
OnRestoreUserInterfaceForPictureInPictureStop	Event called when picture in picture is stopping and is restored to the original user interface.
OnShouldAutomaticallyDismissAtPictureAndPictureStart	Event called when picture in picture is starting and asks if the player view controller should automatically dismiss.
OnWillStartPictureInPicture	Event called when picture in picture will start.
OnWillStopPictureInPicture	Event called when picture in picture will stop.



## 2.53.5 Picture in Picture (iOS 9)

To support picture in picture, there are 2 additional steps that need to be taken.

1. An additional entry is needed inside the plist. The `UIBackgroundModes` property in the Project Options -> Version Info needs to contain the audio entry.

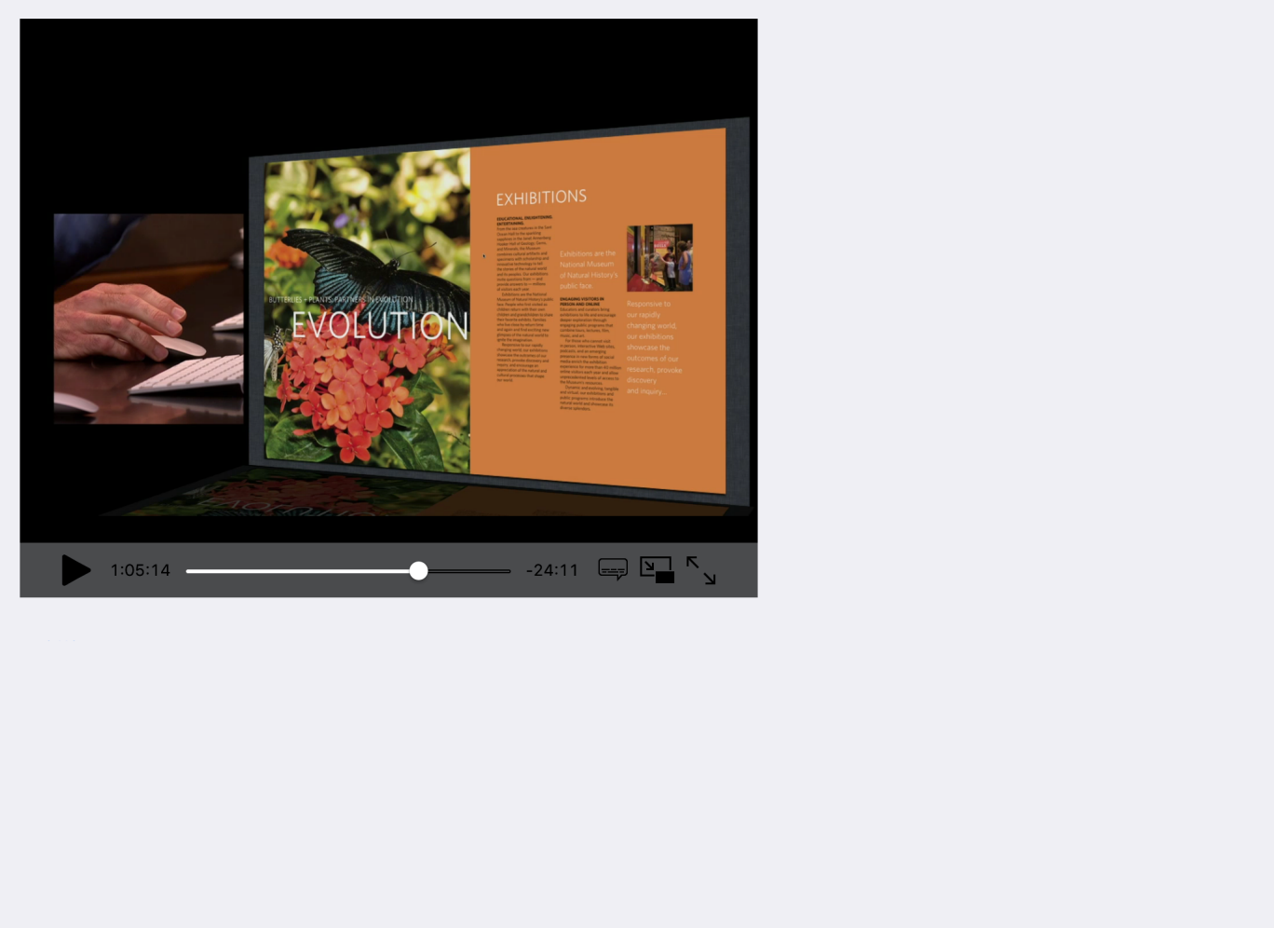


When building the application, the audio entry will be added and the application will then be able to support picture in picture playback mode.

2. Additionally, the audio/video session needs to be initialized with the playback category, needed to support picture in picture. The `TTMSFMXNativeAVPlayerViewController` exposes a class function that needs to be called in the constructor of the form:

```
TTMSFMXNativeAVPlayerViewController.InitializePlayback;
```

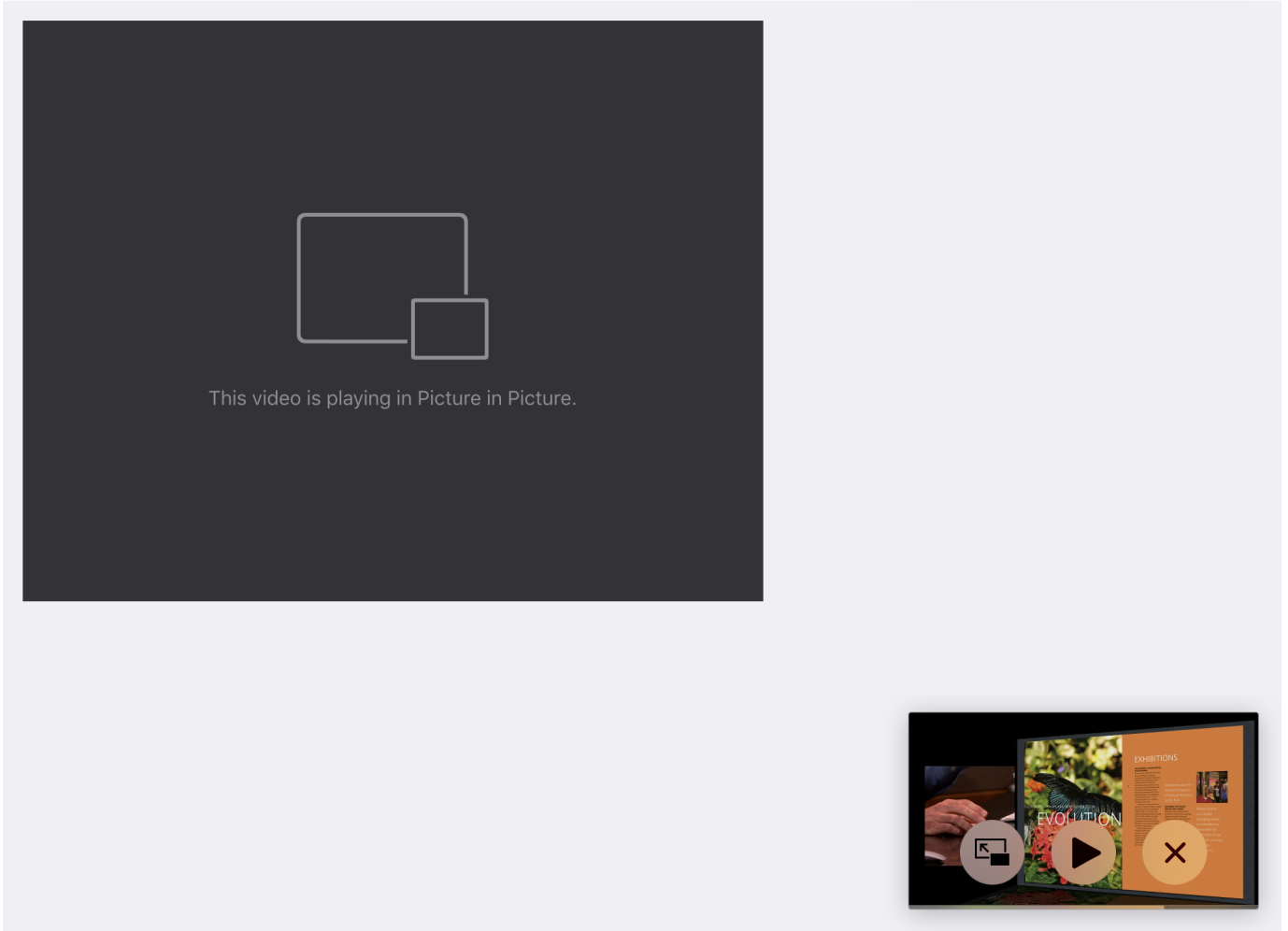
When these steps are successfully executed, the button in the right corner will allow you to display the video outside of the application as demonstrated in the following sample:



No SIM

12:37

100 %

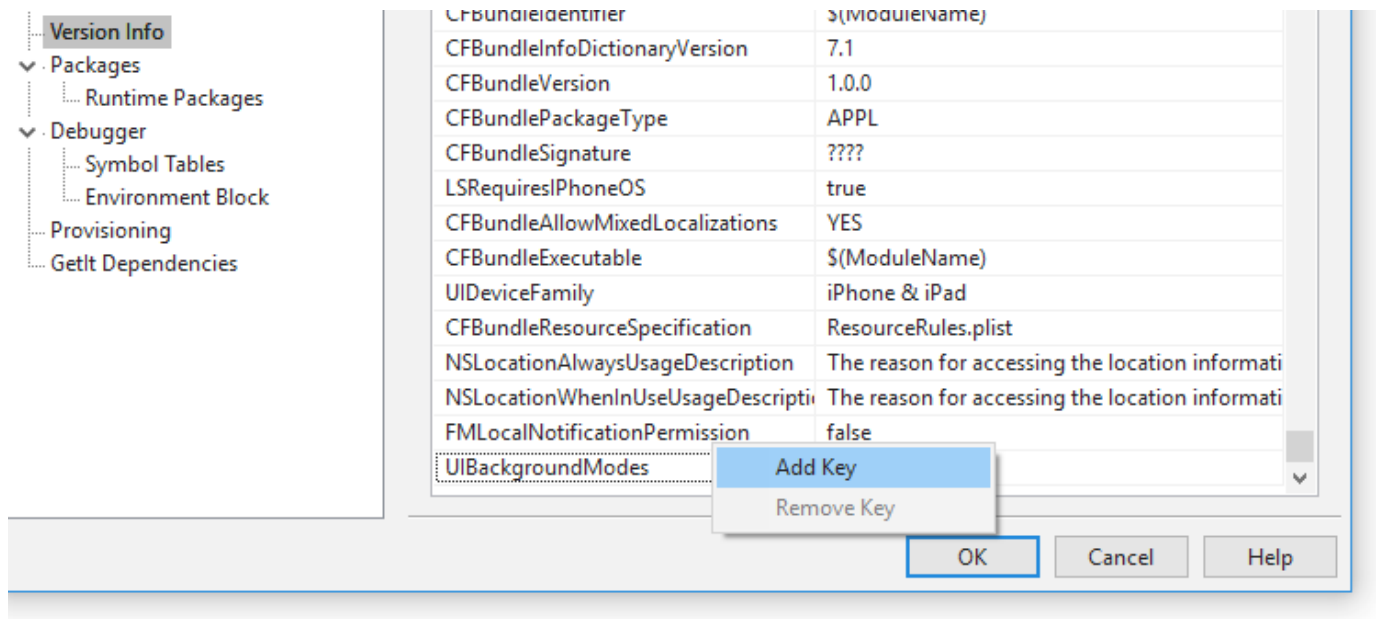


When closing the application, the player will remain on top. The left button in the picture in picture view will allow you to return to the application.

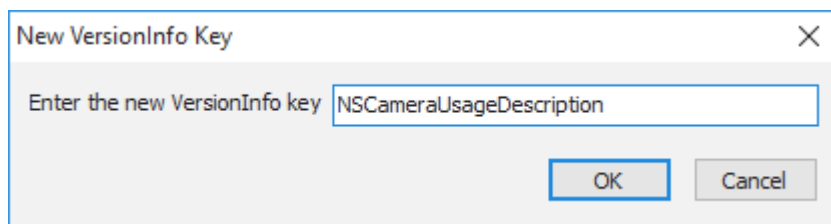
## 2.54 TMSFMXNativeCameraViewController

The `TMSFMXNativeCameraViewController` is a view controller that is capable of rendering a preview of the input of a camera device. After dropping the camera component on the form, a few initialization steps are necessary:

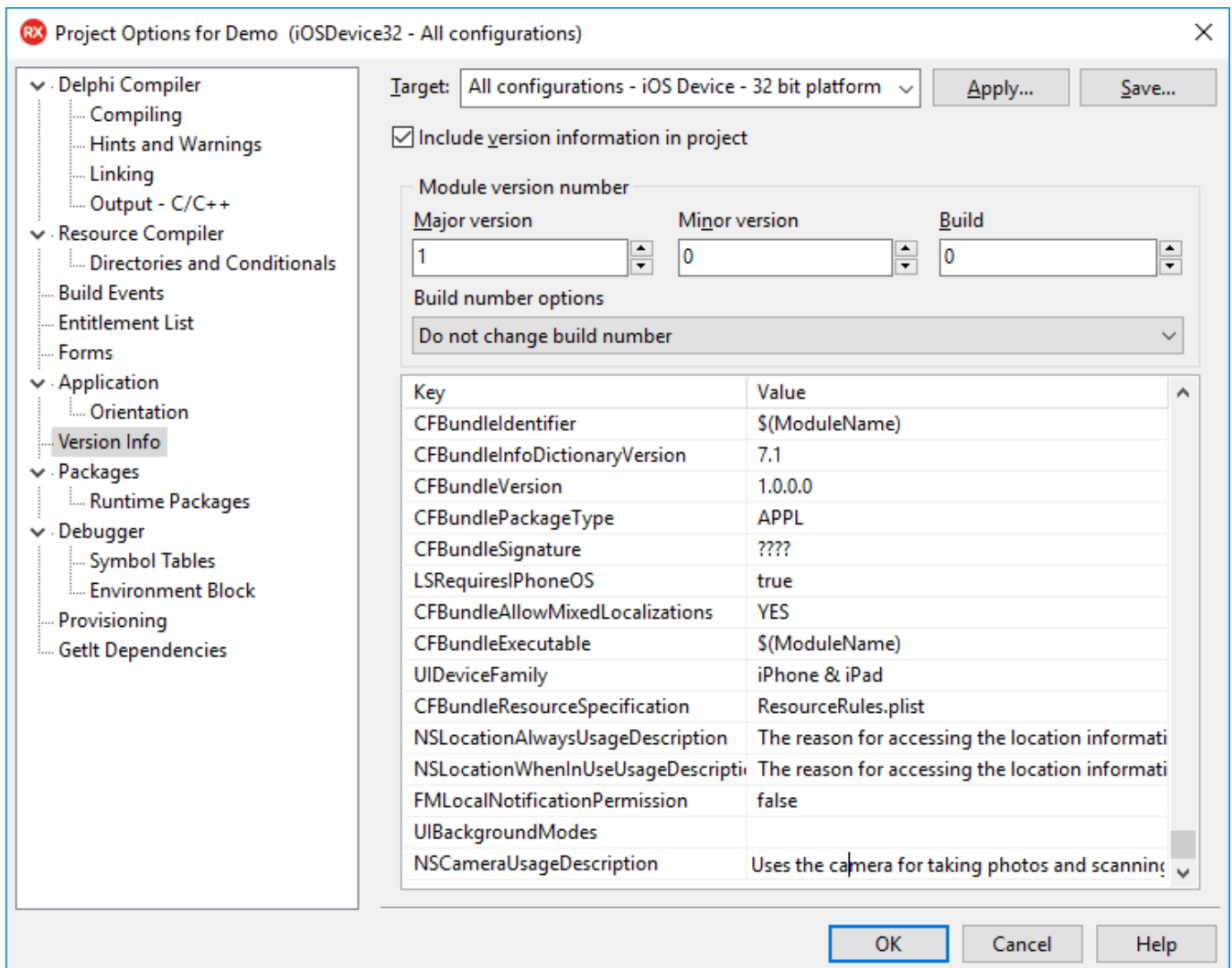
Starting from iOS 10 a new `NSCameraUsageDescription` key is necessary in order to correctly initialize the camera and prevent the application from crashing. This key needs to be added to each individual project. Start by going to the project options and go to version info. Scroll to the bottom, right-click and select “Add Key”.



A dialog will popup, prompting for the new version info key. Fill in “NSCameraUsageDescription”.



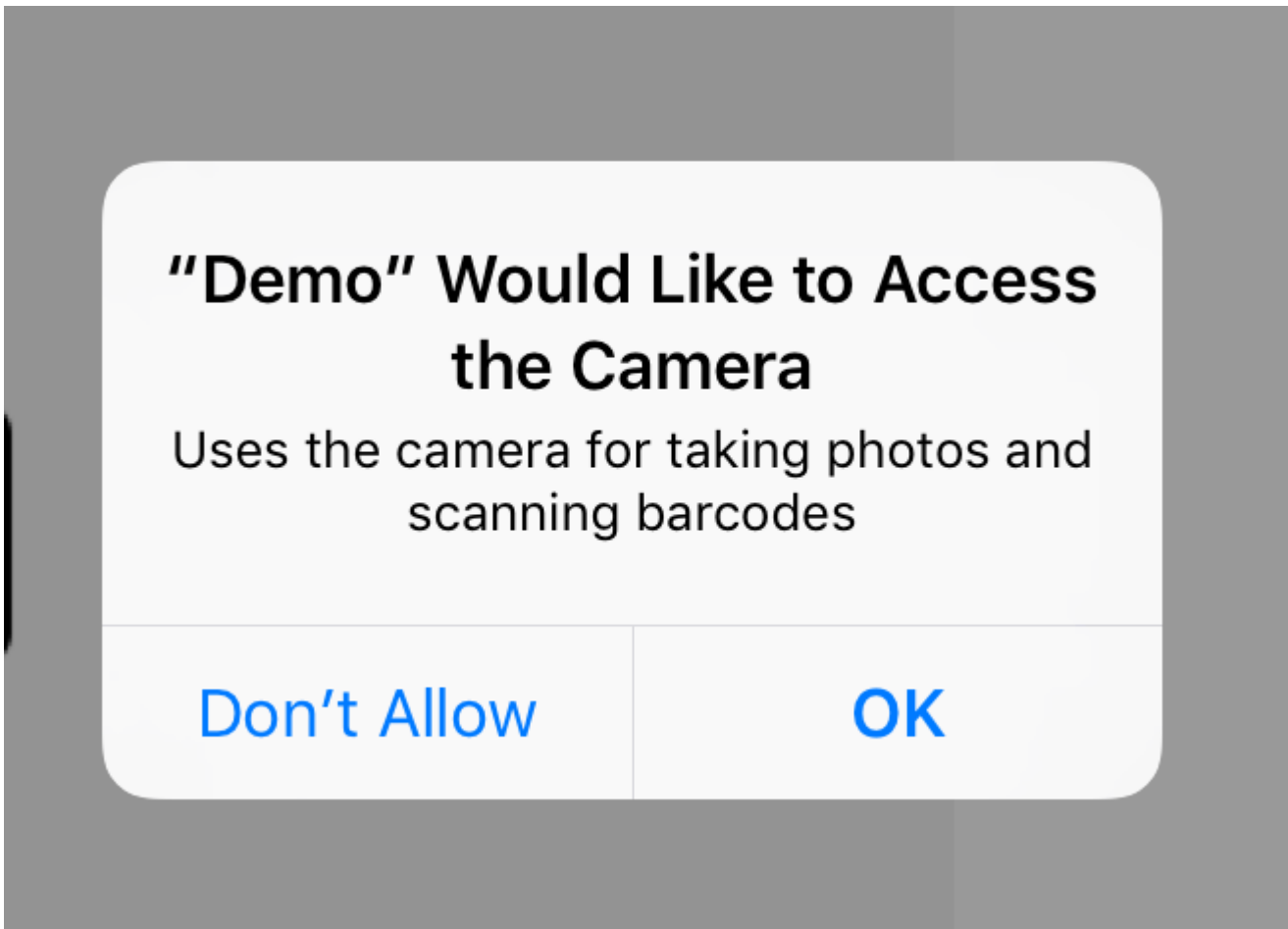
After clicking “OK”, the new entry still needs a value, which can be anything descriptive for your application. In the demo, we have added “Uses the camera for taking photos and scanning barcodes”



After configuring the project, the camera needs to be initialized. This is done via the `InitializeCamera` call:

```
TMSFMXNativeCameraViewController1.InitializeCamera(True);
```

The additional parameter determines if the camera preview can be started or not. Initialization is only necessary once during the lifetime of an application. The authorization based on the `NSCameraUsageDescription` (iOS 10 and newer) will prompt for camera access. As soon as the application has been granted camera access the application has a correctly initialized camera. After initialization, the `OnInitialized` event is called. If for some reason initialization fails, or the user has not granted permission to access the camera, a dialog is shown during authentication to determine if the user still wants to access the camera.



In the popup version, the initialization is automatically started. To start the popup, call one of the Show\* methods that can be popped up from on a native `UIBarButtonItem` or a `TControl`. On iPad, a popup can be shown that covers a section of the screen. The `PopupWidth` and `PopupHeight` properties can be used to set the width and height of the popup. On iPhone, all calls will automatically show the popup full screen, as popups are not supported on iPhone devices.

When taking a photo programmatically with `CapturePhoto` or with the capture photo button in the popup version, the `OnCapturePhoto` event will be triggered. Additional events are available to speed up the process of capturing a photo (`OnCapturePhotoData`, `OnCapturePhotoStream`).

## 2.55 TMSFMXNativeBarCodeScanner

---

The `TMSFMXNativeBarCodeScanner` component is a component that inherits from `TMSFMXNativeCameraViewController` and also has a popup version. The bar code scanner will automatically scan for the supported codes that are stored inside the `SupportedCodes` property. The supported codes are: UPCE, Code39, Code39Mod43, EAN13, EAN8, Code93, Code128, PDF417, QR, Aztec, Interleaved2of5, ITF14, DataMatrix. By default all codes are supported, but the `SupportedCodes` property set can be configured to only allow a set of codes. If a code is detected, the `OnCaptureCode` event is triggered and the `TMSFMXNativeBarCodeScanner` is automatically stopped. This is to prevent that multiple codes are scanned and processed. After each succesful capture, the `TMSFMXNativeBarCodeScanner` needs to be restarted. This can be done with `TMSFMXNativeBarCodeScanner1.Start`. The popup version automatically starts and stops scanning, and also automatically closes when a bar code has been detected.



## 2.56 TMSFMXNativeAppShortcuts

---

### 2.56.1 Overview

---

#### Usage

The `TMSFMXNativeAppShortcuts` class adds different shortcuts with linked actions by pressing your app's icon on the home screen.

#### Public Methods

Methods name	Description
<code>ClearShortcutItems</code>	Clears the list of shortcut items on the iOS device.
<code>UpdateShortcutItems</code>	Pushes the list of shortcut items on the iOS device.

#### Published Events

Events name	Description
<code>OnShortcutItemsUpdated</code>	Event called when the list of shortcut items is pushed on the iOS device.
<code>OnShortcutItemExecuted</code>	Event called when the a shortcut item was executed.
<code>OnShortcutItemNotFound</code>	Event called when the app was opened via a shortcut item, but the item couldn't be found.



## 2.56.2 Properties

---

### Overview

#### PUBLISHED PROPERTIES

Property name	Description
ShortcutItems	Collection of <a href="#">TMSFMXNativeAppShortcutItem</a> .

---

## ShortcutItem

### OVERVIEW

#### Usage

The `TMSFMXNativeAppShortcutItem` is the collection item used in `TMSFMXNativeAppShortcuts`.

#### Public Properties

Property name	Description
Data	Object that can be used for additional information.
DataSource	String that can be used for additional information.

#### Published Properties

Property name	Description
Active	Add the shortcut item to application.
Icon	Optional icon ( <code>TMSFMXNativeAppShortcutIconType</code> ) default none.
Subtitle	User-visible subtitle.
Title	User-visible title.
Type	App-specific string that you employ to identify the type of quick action to perform.

#### Published Events

Events name	Description
OnChanged	Event called when the properties of the shortcut item are changed.
OnExecute	Event called when the application is opened via this item.

[Go back to Properties](#)

## SHORTCUTICONTYPE

### Usage

The `TMSFMXNativeAppShortcutIconType` is an enumerator for the `Icon` property of `TMSFMXNativeAppShortcutItem`.

Values

**Possible values**

---

asiNone

---

asiAdd

---

asiAlarm

---

asiAudio

---

asiBookmark

---

asiCapturePhoto

---

asiCaptureVideo

---

asiCloud

---

asiCompleted

---

asiCompose

---

asiConfirmation

---

asiContact

---

asiDate

---

asiFavorite

---

asiHome

---

asiInvitation

---

asiLocation

---

asiLove

---

asiMail

---

asiMarkLocation

---

asiMessage

---

asiPause

---

asiPlay

---

asiProhibited

---

asiSearch

---

asiShare

---

asiShuffle

---

asiTask

---

**Possible values**

asiTime

---

asiUpdate

---

[Go back to ShortcutItem](#)

## 2.57 TMSFMXNativeSpeechRecognition

---

### 2.57.1 Usage

With `TMSFMXNativeSpeechRecognition` you can transcribe captured audio and recordings to written text.

### 2.57.2 Published Properties

Property name	Description
StopAfterPause	Stop recording after a pause is detected.
RestartAfterPause	Restart the transcription after a pause is detected.
PauseInterval	The interval in ms to trigger when no changes are in the result.
ContextualStrings	List of strings that are uncommon but should be tried to match in the text.
RetrievePartialResults	If false will only trigger the result event when the task is finished or recording stopped.
Locale	String which indicates the locale to transcribe the speech or audio file.

### 2.57.3 Public Properties

Property name	Description
AudioEngine	The internal <code>AVAudioEngine</code> .
InputNode	The internal <code>AVAudioEngine</code> input node.
IsRecording	Indicates if the microphone is currently recording.
IsTranscribing	Indicates if the speech recognizer is currently transcribing.
MicRequest	The internal <code>SFSpeechAudioBufferRecognitionRequest</code> connected to the <code>SpeechRecognizer</code> .
PermissionStatus	Indicates if the speech recognition is authorized, after the <code>RecognitionPermissionRequest</code> method.
RequiresOnDeviceRecognition	A Boolean value that determines whether a request must keep its audio data on the device if supported.
SpeechRecognizer	The internal <code>SFSpeechRecognizer</code> .
SpeechRecognizerAvailable	A Boolean value that indicates whether the speech recognizer is currently available.
SupportedLocales	List of the different language settings that are supported on the device.

## 2.57.4 Public Methods

Methods name	Description
FinishRequest	Finishes the request to transcribe an audio file.
RecognitionPermissionRequest	Asks the first time for the user's permission to perform speech recognition using Apple's servers.
StopRecording	Stops the recording via the microphone.
TranscribeFromFile(APath: string)	Starts the request to transcribe an audio file.
TranscribeFromMicrophone	Starts the recording of the microphone and transcribes the input.

## 2.57.5 Published Events

Events name	Description
OnPermissionResult	Event called when the permission request is handled.
OnTranscribeFileResult	Event called when a transcription is ready of the audio file.
OnTranscribeMicrophoneResult	Event called when a transcription is ready of microphone recording.
OnTranscribeError	Event called when an error is thrown.
OnTranscribeFileStarted	Event called when a transcription of an audio file is started.
OnTranscribeMicrophoneStarted	Event called when a transcription of a microphone recording is started.
OnTranscribePause	Event called when a pause is detected in the microphone recording.



## 2.58 TMSFMXNativeSpeechCommandRecognition

---

### 2.58.1 Overview

#### Usage

With `TMSFMXNativeSpeechCommandRecognition` you can transcribe captured audio and recordings to written text.

#### Public Properties

Property name	Description
PermissionStatus	Indicates if the speech recognition is authorized, after the <code>RecognitionPermissionRequest</code> method.
SupportedLocales	List of the different language settings that are supported on the device.

#### Public Methods

Methods name	Description
FinishRequest	Finishes the request to transcribe an audio file.
RecognitionPermissionRequest	Asks the first time for the user's permission to perform speech recognition using Apple's servers.
StopRecording	Stops the recording via the microphone.
TranscribeFromMicrophone	Starts the recording of the microphone and transcribes the input.

#### Published Events

Events name	Description
OnPermissionResult	Event called when the permission request is handled.
OnTranscribeMicrophoneResult	Event called when a transcription is ready of microphone recording.
OnTranscribeError	Event called when an error is thrown.
OnTranscribeMicrophoneStarted	Event called when a transcription of a microphone recording is started.
OnTranscribePause	Event called when a pause is detected in the microphone recording.

## 2.58.2 Properties

---

### Overview

#### PUBLISHED PROPERTIES

Property name	Description
Commands	Collection of <a href="#">TTMSFMXNativeSpeechCommand</a> .
StopAfterPause	Stop recording after a pause is detected.
PauseInterval	The interval in ms to trigger when no changes are in the result.
ContextualStrings	List of strings that are uncommon but should be tried to match in the text.
Locale	String which indicates the locale to transcribe the speech or audio file.

## ShortcutItem

### OVERVIEW

#### Usage

The `TMSFMXNativeSpeechCommand` is the collection item used in `TMSFMXNativeSpeechCommandRecognition`.

#### Configure your command

The text will be converted to a regular expression internally. You can use three different special characters to get more out of your command:

- With the asterisk you can indicate that you want to retrieve the rest of the text as a parameter e.g. `Get me *things .`
- With the colon you can retrieve one word as a parameter e.g. `:Do for me .`
- With the brackets you can add it as a possible part of the text e.g. `Hello (there) phone .`

#### Published Properties

Property name	Description
Name	Id of the command.
Text	The text to check for.
IsRegex	Set to true if the text is already a regular expression.
StartsWithText	The partial result should start with the text.
EndsWithText	The partial result should end with the text.

#### Published Events

Events name	Description
OnCommand	Event called when the text is detected in the transcription.

[Go back to Properties](#)

## 2.59 TMSFMXNativeWKWebView

---

### 2.59.1 Usage

The `TMSFMXNativeWKWebView` class defines a view that you use to incorporate web content seamlessly into your app's UI.

### 2.59.2 Public Properties

Property name	Description
WebView	Returns a reference to the native iOS <code>WKWebView</code> .

### 2.59.3 Public Methods

Method name	Description
CanGoBack: Boolean	Returns a boolean if the WebView can go back a page.
CanGoForward: Boolean	Returns a boolean if the WebView can go forward a page.
ExecuteJavaScript(AScript: String): String	Executes JavaScript on the current page.
GetView	Returns a reference to the native iOS <code>UIView</code> .
GoBack	Goes back one page.
GoForward	Goes forward one page.
isLoading	Returns a boolean whether the WebView is loading or not.
LoadFile(AFile: String)	Loads a specific file in the WebView.
LoadHTMLString(AHTML: String)	Loads a specific HTML string or HTML content in the WebView.
Navigate(AUrl: String)	Navigates to a specific URL.
Reload	Reloads the current page.
StopLoading	Stops loading the current page.

## 2.59.4 Published Events

---

<b>Event name</b>	<b>Description</b>
OnDidFailLoadWithError	Event that is called when the loading failed.
OnDidFinishLoad	Event that is called when the loading is finished.
OnDidStartLoad	Event that is called when the loading is started.
OnShouldStartLoadWithRequest	Event that is called when the WebView should start loading with a specific request.

---