
# TMS mCL
# DEVELOPERS GUIDE

July 2020


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

# Index

# Availability

TMS mCL is a set of components for true native Mac application development and is available for Embarcadero Delphi XE11 & C++Builder XE11 or newer releases.

# Online references

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

TMS mCL page:
https://www.tmssoftware.com/site/tmsmCL.asp

Mac Developer Library:
https://developer.apple.com/library/mac/navigation/

# List of included components

- TMSFMXNativeNSView
- TMSFMXNativeNSButton
- TMSFMXNativeNSTextField
- TMSFMXNativeNSSecureTextField
- TMSFMXNativeNSLabel
- TMSFMXNativeNSLevelIndicator
- TMSFMXNativeNSProgressIndicator
- TMSFMXNativeNSTabView
- TMSFMXNativeNSPopover
- TMSFMXNativeNSToolbar
- TMSFMXNativeNSDatePicker
- TMSFMXNativeNSComboBox
- TMSFMXNativePDFView
- TMSFMXNativeNSStepper
- TMSFMXNativeNSRadioButton
- TMSFMXNativeNSCheckBox
- TMSFMXNativeNSTextView
- TMSFMXNativeNSRichTextView
- TMSFMXNativeNSRichTextViewToolBar
- TMSFMXNativePDFThumbnailView
- TMSFMXNativeNSScrollView
- TMSFMXNativeWebView
- TMSFMXNativeNSSlider
- TMSFMXNativeNSImageView
- TMSFMXNativeNSTableView
- TMSFMXNativeNSOutlineView
- TMSFMXNativeMaciCloud
- TMSFMXNativeMaciCloudDocument
- TMSFMXNativeMacPDFLib
- TMSFMXNativeNSPopupButton

# TMSFMXNativeNSView


**Usage**

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

**Published Properties**

| | |
|---|---|
| Color | The background color of the view. |

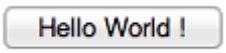
**Public Properties**

| | |
|---|---|
| View | Returns a reference to the native Mac NSView. |

**Events**

| | |
|---|---|
| OnDrawRect | Event to perform custom drawing inside the View. |

# TMSFMXNativeNSButton




**Usage**

An instance of TMSFMXNativeNSButton shows a native Mac Button on the screen.

**Published Properties**

| | |
|---|---|
| AllowsMixedState | Property to allow mixed state in Radio and Check button mode. |
| BezelStyle | Various button bezel styles to change the appearance. |
| Bitmap | Property used to show a bitmap on the Button. |
| Font | The font of the button text. |
| State | The state of the button for radio and check button mode. |
| Style | The style of the Button. The Button style can be set to one of the following values:<br>bsMomentaryChangeButton<br>bsMomentaryLight<br>bsMomentaryLightButton (**Default**)<br>bsMomentaryPushButton<br>bsMomentaryPushInButton<br>bsOnOffButton<br>bsPushOnPushOffButton<br>bsRadioButton<br>bsSwitchButton<br>bsToggleButton |
| Text | The text of the button. |

**Public Properties**

| | |
|---|---|
| Button | Returns a reference to the native Mac NSButton. |

**Events**

| | |
|---|---|
| OnClick | Event called when clicking on the Button. |

# TMSFMXNativeNSPopupButton


**Usage**

An instance of TMSFMXNativeNSPopupButton shows a native Mac Popup button on the screen and implements a pop-up and pull-down menu that displays items.

**Published Properties**

| | |
|---|---|
| Items | The items to show in the pop-up or pull-down menu. |
| SelectedItemIndex | The selected item index. |
| Font | The font of the button text. |
| PullsDown | When False, uses a pop-up menu. When true, uses a pull-down menu. |
| Title | The title of the button. |

**Public Properties**

| | |
|---|---|
| Button | Returns a reference to the native Mac NSPopupButton. |
| SelectedText | Gets or sets the text on the popup button and selects the correct item. |

**Events**

| | |
|---|---|
| OnSelectionChanged | Event called when the selection changes on the popup button. |

# TMSFMXNativeNSTextField / TMSFMXNativeNSSecureTextField


**Usage**

A TMSFMXNativeNSTextField 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. The TMSFMXNativeNSSecureTextField has the same properties and functionality as the TMSFMXNativeNSTextField but hides the text from the user by displaying other characters. This control is typically used as a password-entry.

**Published Properties**

| | |
|---|---|
| Bordered | Draws border around the TextField, optionally. |
| DrawsBackGround | Shows or hides the background of the TextField. |
| Editable | Enables or disables editing. |
| Font | The font of the text. |
| Selectable | Enables or disables selection of the text. |

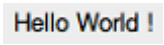
**Public Properties**

| | |
|---|---|
| TextField | Returns a reference to the native Mac NSTextField. |

**Events**

| | |
|---|---|
| OnChange | Event called when the text is changing in the TextField. |
| OnDidBegin | Event called when editing did begin in the TextField. |
| OnDidEnd | Event called when editing did end in the TextField. |

# TMSFMXNativeNSLabel




**Usage**

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

**Published Properties**

| | |
|---|---|
| Bordered | Draws border around the Label, optionally. |
| DrawsBackGround | Shows or hides the background of the Label. |
| Editable | Enables or disables editing. |
| Font | Specifies the font name and size of the Label. |
| Selectable | Enables or disables selection of the text. |
| Text | The Text of the Label. |

**Public Properties**

| | |
|---|---|
| Lbl | Returns a reference to the native Mac NSLabel. |

**Events**

| | |
|---|---|
| OnChange | Event called when the text is changing in the Label. |
| OnDidBegin | Event called when editing did begin in the Label. |
| OnDidEnd | Event called when editing did end in the Label. |

# TMSFMXNativeNSLevelIndicator

**Usage**

Level indicators provide a visual representation of a level or amount of something, using discrete values.

**Published Properties**

| | |
|---|---|
| CriticalValue | The value that indicates the level indicator current value is at a critical level. The level indicator color is red at that value. |
| CurrentValue | The current value of the indicator. When the value of the level indicator is below the warning and critical value the indicator color is green. |
| MaxValue | The maximum value of the indicator. |
| MinValue | The minimum value of the indicator. |
| NumberOfMajorTickMarks | Number of major tick marks. |
| NumberOfTickMarks | Number of minor tick marks. |
| TickMarkPosition | Position of the tickmarks on the level indicator. |
| WarningValue | The value that indicators the level indicator current value is at a warning level. The level indicator color is green at that value. |

**Public Properties**

| | |
|---|---|
| LevelIndicator | Returns a reference to the native Mac NSLevelIndicator. |

**Events**

| | |
|---|---|
| OnValueChanged | Event called when value changed. |

# TMSFMXNativeNSProgressIndicator

**Usage**

The TMSFMXNativeNSProgressIndicator lets an application display a progress indicator to show that a lengthy task is under way.

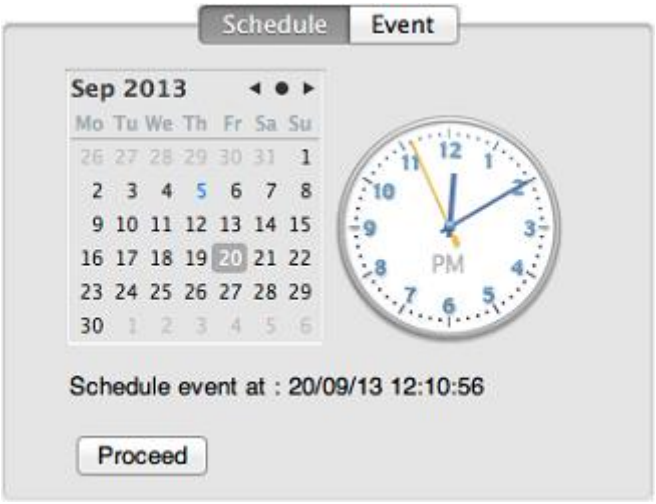
**Published Properties**

| | |
|---|---|
| CurrentValue | The current value of the Progress Indicator |
| Indeterminate | Shows the Progress Indicator with an infinite animation loop. |
| MaxValue | The maximum value of the Progress Indicator |
| MinValue | The minimum value of the Progress Indicator |
| Style | The style of the ProgressIndicator<br><br>isProgressIndicatorBarStyle (**Default**)<br><br>isProgressIndicatorSpinningStyle |
| UsesThreadedAnimation | Enables threaded animation which makes sure that the progress animation keeps running even when there is a huge load in the main thread. |

**Public Properties**

| | |
|---|---|
| ProgressIndicator | Returns a reference to the native Mac NSProgressIndicator. |

# TMSFMXNativeNSTabView




**Usage**

A TMSFMXNativeNSTabView provides a convenient way to present information in multiple pages.

**Published Properties**

| | |
|---|---|
| AllowsTruncatedLabels | Allows display of truncated labels when the tabs do not have enough space available to display its title. |
| ControlTint | A set of operating system defined colors for the TabView. |
| DrawsBackGround | Enables / disables drawing of the background. |
| SelectedItem | The selected item / selected tab |
| SelectedItemIndex | The index of the selected item / selected tab. |
| TabType | The various tab types that can be used to display tabs at a different location with a different appearance.<br><br>ttBottomTabsBezelBorder<br><br>ttLeftTabsBezelBorder<br><br>ttNoTabsBezelBorder<br><br>ttNoTabsLineBorder |

| | |
|---|---|
| | ttNoabsNoBorder<br><br>ttRightTabsBezelBorder<br><br>ttTopTabsBezelBorder (**Default**) |

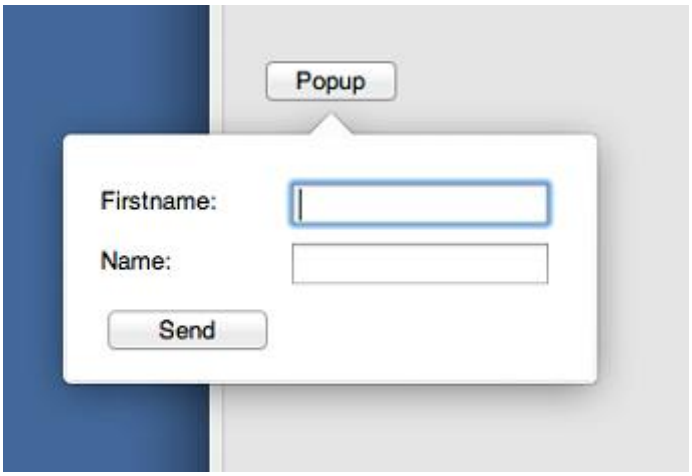
**Public Properties**

| | |
|---|---|
| TabView | Returns a reference to the native Mac NSTabView. |

**Events**

| | |
|---|---|
| OnItemChanged | Event called when an item (tab) has changed. |
| OnItemWillChange | Event called when an item (tab) will change. |

# TMSFMXNativeNSPopover




**Usage**

The TMSFMXNativeNSPopover class provides a means to display additional content related to existing content on the screen. The popover is positioned relative to the existing content

**Published Properties**

| | |
|---|---|
| Animates | Enable / Disable animation when showing the popup. |
| View | The view that needs to be displayed inside the popup. |
| WindowView | The windowview that needs to be displayed when dragging and transitioning from popup to window view. |

**Public Properties**

| | |
|---|---|
| Popover | Returns a reference to the native Mac NSPopover. |

**Events**

| | |
|---|---|
| OnDidClose | Event called when the popup did close. |
| OnDidShow | Event called when the popup did show. |

| | |
|---|---|
| OnShouldClose | Event called when the popup should close. |
| OnWillClose | Event called when the popup will close. |
| OnWillShow | Event called when the popup will show. |

# TMSFMXNativeNSToolbar


**Usage**

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

**Published Properties**

| | |
|---|---|
| AllowsUserCustomization | Allows the user to customize the toolbar, add / remove items from it, or change the appearance. |
| AutoSavesConfiguration | Automatically saves the configuration for the next use. |
| DisplayMode | The display mode of the ToolBar. |
| Items | The items displayed on the ToolBar. |
| Items[Index] → Action | Property to assign an action combined with an action list. |
| Items[Index] → Bitmap | The bitmap used inside an item. |
| Items[Index] → Enabled | Enables / disables an item. |
| Items[Index] → Kind | The kind of item that is display in the ToolBar, an item can be a normal, system or custom item. |
| Items[Index] → Option | Shows the item optionally in a separate dialog that allows dragging of that item to the toolbar. Items can be placed by default on the toolbar or in the allowed items dialog. |
| Items[Index] → SystemItem | When setting the kind to ikSystem, the SystemItem property determines which icon is display inside the button. |
| Items[Index] → Text | The text of a button. |
| Items[Index] → Visible | Shows / hides a button. |
| ShowBaseLineSeparator | Returns a Boolean value that indicates whether |

| | |
|---|---|
| | the toolbar shows the separator between the toolbar and the main window contents. |
| SizeMode | The size mode of the ToolBar, which defines if icon, text or both are displayed. |

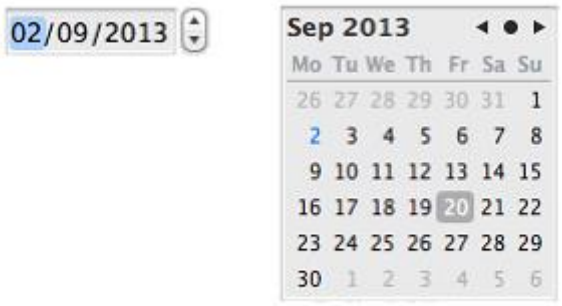
**Public Properties**

| | |
|---|---|
| ToolBar | Returns a reference to the native Mac NSToolbar. |

**Events**

| | |
|---|---|
| OnItemClick | Event called when clicking on an item (button) |
| OnColorChanged | Event called when the color in the color panel changes. The color panel can be called from the toolbar when a system item toolbar button is added. |

# TMSFMXNativeNSDatePicker




**Usage**

TMSFMXNativeNSDatePicker is a control that provides a user interface for displaying and editing a datetime.

**Published Properties**

| | |
|---|---|
| DateTime | The datetime displayed by the DatePicker. |
| Elements | Set of elements that define which parts of a datetime are displayed in the DatePicker. |
| EndDateTime | The End datetime when a range of dates are selected. |
| MaximumDateTime | The maximum datetime that a DatePicker can show. |
| MinimumDateTime | The minimum datetime that a DatePicker can show. |
| Mode | The mode of the DatePicker that allows switching between single and range mode. |
| Style | The style of the DatePicker. |

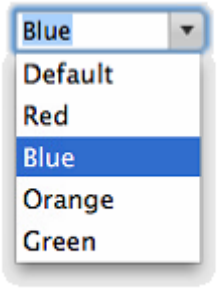
**Public Properties**

| | |
|---|---|
| DatePicker | Returns a reference to the native Mac NSDatePicker. |

**Events**

| | |
|---|---|
| OnDateChanged | Event called when a date is selected. |
| OnDateRangeChanged | Event called when a date range is selected. |

# TMSFMXNativeNSComboBox




**Usage**

A TMSFMXNativeNSComboBox is a control that allows you to either enter text directly (as you would with a TMSFMXNativeNSTextField) or click the attached arrow at the right of the combo box and select from a displayed ("pop-up") list of items.

**Published Properties**

| | |
|---|---|
| Editable | Enables editing of the values. |
| ItemIndex | The current selected item. |
| Items | A TStringList of items displayed in the ComboBox. |

**Public Properties**

| | |
|---|---|
| ComboBox | Returns a reference to the native Mac NSComboBox. |

**Events**

| | |
|---|---|
| OnChange | Event called when the value is changed. |
| OnChanging | Event called when the value is changing. |
| OnClose | Event called when the popup is closed. |
| OnGetNumberOfItems | Event called to get the number of items in a combobox. |
| OnGetValueForItems | Event called to get the value for the items in a combobox. |

| | |
|---|---|
| OnShow | Event called when the popup is shown. |

# TMSFMXNativePDFView


**Usage**

A TMSFMXNativePDFView encapsulates the functionality of PDF Kit, to view and navigate through PDF files.

**Published Properties**

| | |
|---|---|
| BackgroundColor | The backgroundcolor of the PDFView. |
| DisplayMode | The displaymode of the PDFView. Can display in multiple pages, optionally continuous. |
| DisplaysAsBook | Returns a Boolean value indicating whether the view will display the first page as a book cover. |
| DisplaysPageBreaks | Toggles the display of page breaks. |
| Location | The location of the PDF file. |
| PageIndex | The current page index of the PDF. |
| ShouldAntiAlias | Boolean whether the PDFView will draw its pages with or without antialias. |

**Public Properties**

| | |
|---|---|
| PDFView | Returns a reference to the native Mac PDFView. |

**Events**

| | |
|---|---|
| OnPageChanged | Event called when the page has changed. |

# TMSFMXNativeNSStepper

**Usage**

A TMSFMXNativeNSStepper control provides a user interface for incrementing or decrementing a value.

**Published Properties**

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

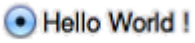
**Public Properties**

| | |
|---|---|
| Stepper | Returns a reference to the native Mac NSStepper. |

**Events**

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

# TMSFMXNativeNSRadioButton




**Usage**

Inherits from TMSFMXNativeNSButton and presets the style to be a radiobutton.

**Published Properties**

| | |
|---|---|
| Checked | The checked property of the radiobutton. |

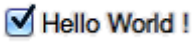
**Public Properties**

| | |
|---|---|
| Button | Returns a reference to the native Mac NSButton. |

**Events**

| | |
|---|---|
| OnClick | Event called when the radiobutton is clicked. |

# TMSFMXNativeNSCheckBox




**Usage**

Inherits from TMSFMXNativeNSButton and presets the style to be a checkbox.

**Published Properties**

| | |
|---|---|
| Checked | The checked property of the checkbox. |

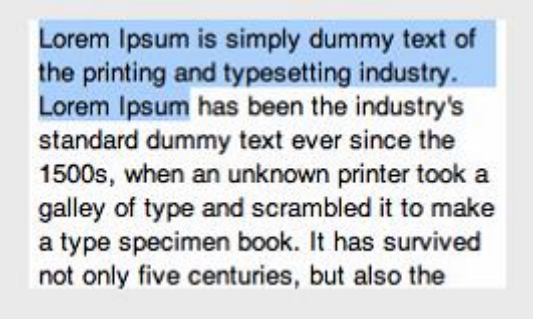
**Public Properties**

| | |
|---|---|
| Button | Returns a reference to the native Mac NSButton. |

**Events**

| | |
|---|---|
| OnClick | Event called when the checkbox is clicked. |

# TMSFMXNativeNSTextView




**Usage**

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

**Published Properties**

| | |
|---|---|
| DrawsBackGround | A Boolean value indicating whether the background is drawn or not. |
| Editable | A Boolean value indicating whether the TextView is editable or not. |
| Font | Specifies the Font name and Size of the TextView. |
| Selectable | Enables / disables selection of the text. |
| Text | The Text of the TextView. |

**Public Properties**

| | |
|---|---|
| TextView | Returns a reference to the native Mac NSTextView. |

**Events**

| | |
|---|---|
| OnChange | Event called when the text value is changed. |
| OnDidBegin | Event called when the text editing did begin. |

| | |
|---|---|
| OnDidEnd | Event called when the text editing did end. |

# TMSFMXNativePDFThumbnailView

**Usage**

TMSFMXNativePDFThumbnailView contains a set of thumbnails, each of which represents a page in a PDF document.

**Published Properties**

| | |
|---|---|
| PDFView | A reference to an instance of TMSFMXNativePDFView. |

**Public Properties**

| | |
|---|---|
| PDFThumbnailView | Returns a reference to the native Mac PDFThumbnailView. |

# TMSFMXNativeNSScrollView

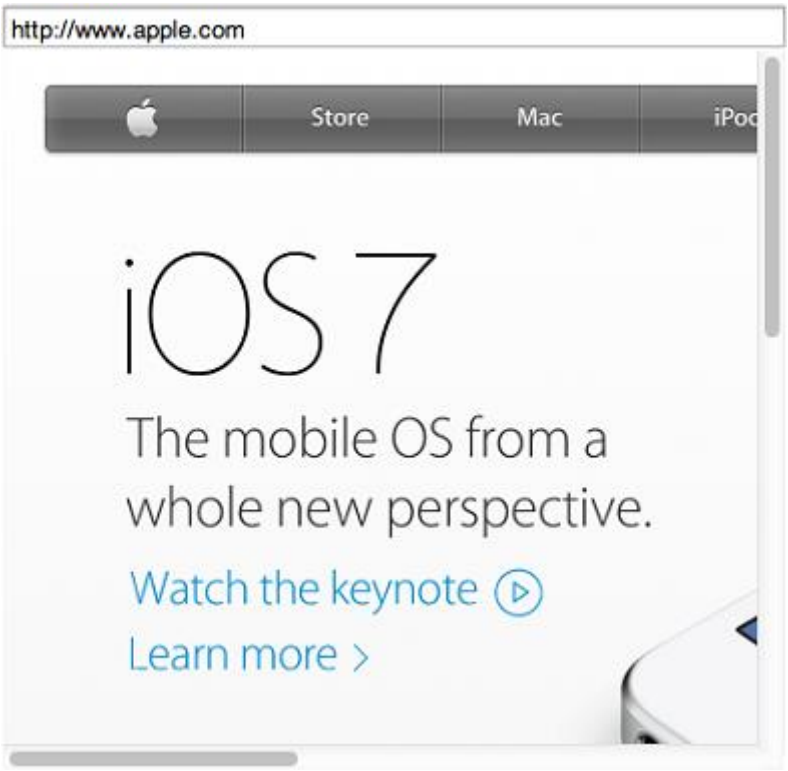

**Usage**

The TMSFMXNativeNSScrollView 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.

**Public Properties**

| | |
|---|---|
| ScrollView | Returns a reference to the native Mac NSScrollView. |

# TMSFMXNativeWebView




**Usage**

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

**Public Properties**

| | |
|---|---|
| WebView | Returns a reference to the native Mac WebView. |

**Published Events**

| | |
|---|---|
| OnBeforeNavigate | Event called before navigation. |
| OnNavigateComplete | Event called when navigation is complete. |

# TMSFMXNativeNSSlider

**Usage**

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

**Published Properties**

| | |
|---|---|
| Continuous | Continuous updates send to the OnValueChanged event. |
| MaximumValue | Contains the maximum value of the Slider. |
| MinimumValue | Contains the minimum value of the Slider. |
| Mode | The mode of the Slider.<br>smCircular<br>smHorizontal (**Default**)<br>smVertical |
| Value | Contains the Slider's current value. |

**Public Properties**

| | |
|---|---|
| Slider | Returns a reference to the native Mac NSSlider. |

**Events**

| | |
|---|---|
| OnValueChanged | Event called when the Slider's value has changed. |

# TMSFMXNativeNSImageView

**Usage**

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

Tagged Image File Format (TIFF)

.tiff, .tif

Joint Photographic Experts Group (JPEG)

.jpg, .jpeg

Graphic Interchange Format (GIF)

.gif

Portable Network Graphic (PNG)

.png

Windows Bitmap Format (DIB)

.bmp, .BMPf

Windows Icon Format

.ico

Windows Cursor

.cur

XWindow bitmap

.xbm

**Properties**

| | |
|---|---|
| Alignment | Alignment of the image inside its frame. |
| AllowsCutCopyPaste | Allows / disallows cut copy and paste of the image. |
| Animates | Animates the image if the image supports animation, else it will only display the first frame. |
| Bitmap | Sets the image of an ImageView. |
| BitmapFile | A direct link to an image file located in the root or documents directory. |
| Editable | Enables / disables dragging the image to a new frame. |
| Scaling | The scaling of the image, with or without aspect ratio, … . |
| URL | Property to load an image from an URL. |

**Methods**

| | |
|---|---|
| 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. |

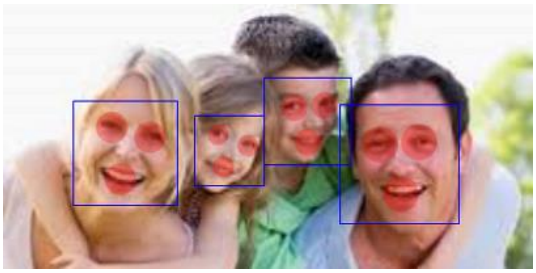
**Public Properties**

| | |
|---|---|
| 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 DetectFaces or ShowFaces. Each face contains information about the position of the left eye, right eye and mouth part and if the eye is closed, the face is smiling and if the face is shown at an angle. |

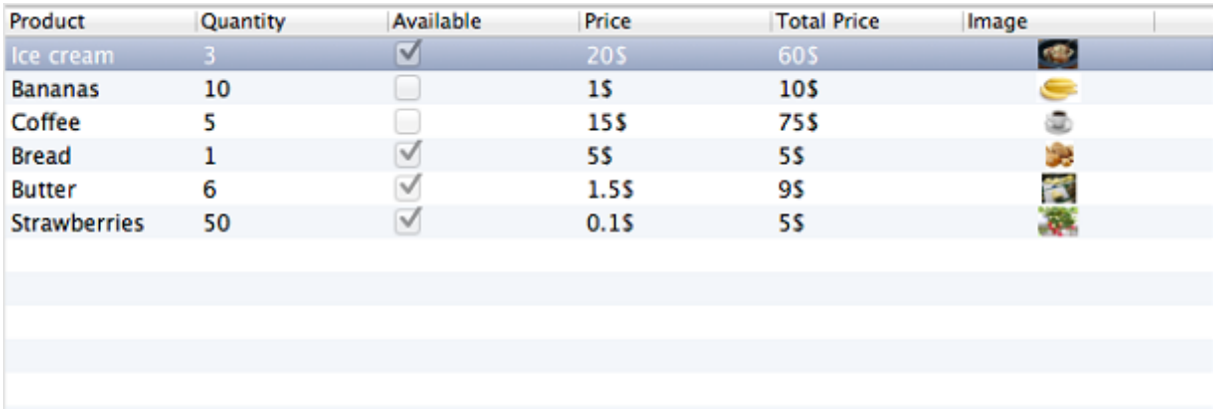
| | |
|---|---|
| ImageView | Returns a reference to the native Mac NSImageView. |
| LeftEyeColor | The color to indicate the left eye after detection of the face. |
| MouthColor | The color to indicate the mouth after detection of the face. |
| RightEyeColor | The color to indicate the right eye after detection of the face. |

**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, detect an eye blink and / or a smile. Below is a sample that demonstrates calling the default ShowFaces call on an image:



# TMSFMXNativeNSTableView




**Usage**

A TMSFMXNativeNSTableView displays data in a columns and rows structure.

**Published Properties**

| | |
|---|---|
| Columns | A collection of columns. |
| Columns[Index] → Alignment | The text alignment for all the cells in a column. |
| Columns[Index] → CellColor | The color for all cells in a column. |
| Columns[Index] → CellTextColor | The text color for all cells in a column. |
| Columns[Index] → Header | The header of a column. |
| Columns[Index] → MaximumWidth | The maximum width of a column. |
| Columns[Index] → MinimumWidth | The minimum width of a column. |
| Columns[Index] → ReadOnly | Enables / disables editing in a column. |
| Columns[Index] → Reordering | Enables / disables reordering of a column. |
| Columns[Index] → ResizingMask | The resizing mask of a column. |
| Columns[Index] → Sorting | Enables / disables sorting of a column. |
| Columns[Index] → Visible | Sets a column visibility. |
| Columns[Index] → Width | The width of a column. |
| Items | A collection of items (rows). |
| Items[Index] → Values | A collection of row values for each column. |

| | |
|---|---|
| Values[Index] → Value | The value of a cell on a specific row and column. The type of this property is TValue and currently supports String, Double, Integer, Boolean and TBitmap. |
| Options | A set of options to configure the TableView. |
| Options → Appearance | The appearance of the TableView. |
| Options → Appearance → AlternatingRowBackgroundColors | Enables / disables the alternating row background colors for the TableView. |
| Options → Appearance → BackgroundColor | The background color of the TableView. |
| Options → Appearance → GridColor | The grid color of the TableView. |
| Options → Appearance → SelectionHighlightStyle | The selection style for the TableView. |
| Options → Appearance → SortIndicator | Enables / disables displaying a sortindicator in the column headers. |
| Options → Interaction | The interaction of the TableView. |
| Options → Interaction → ColumnReordering | Allows column reordering on the TableView. |
| Options → Interaction → ColumnResizing | Allows column resizing on the TableView. |
| Options → Interaction → ColumnSelection | Allows column selection on the TableView. |
| Options → Interaction → EmptySelection | Allows an empty selection in the TableView. |
| Options → Interaction → MultipleSelection | Allows selecting multiple rows in the TableView. |
| Options → Layout | The layout of the TableView. |
| Options → Layout → ColumnAutoResizingStyle | Sets the column auto resizing style for the TableView. |
| Options → Layout → ColumnHeaders | Enables / disables the column header of the TableView. |
| Options → Layout → ColumnSpacing | Spacing between columns. |
| Options → Layout → RowHeight | The row height of the TableView. |
| Options → Layout → RowSizeStyle | The style of the row size of the TableView. |

| | |
|---|---|
| Options → Layout → RowSpacing | The spacing between rows of the TableView. |

**Public Properties**

| | |
|---|---|
| TableView | Returns a reference to the native Mac NSTableView. |

**Events**

| | |
|---|---|
| OnAllowColumnReorder | Event called to determine if a column can be reordered or not. |
| OnCellClick | Event called when a cell is clicked. |
| OnCellCompare | Event called when a cell is compared for sorting. |
| OnCellDblClick | Event called when a cell is double clicked. |
| OnCellEditingCheckBoxClick | Event called when a cell with a checkbox is clicked. |
| OnCellEditingDidBegin | Event called when a cell editing did begin. |
| OnCellEditingDidChange | Event called when a cell editing did change. |
| OnCellEditingDidEnd | Event called when a cell editing did end. |
| OnCellEditingWillChange | Event called when a cell editing will change. |
| OnColumnClick | Event called when a column is clicked. |
| OnColumnDblClick | Event called when a column is double clicked. |
| OnColumnDidDrag | Event called when a column is dragged. |
| OnColumnDidMove | Event called when a column is moved. |
| OnColumnDidResize | Event called when a column is resized. |
| OnGetCellAlignment | Event called to determine the cell text alignment. |
| OnGetCellColor | Event called to determine the cell color. |
| OnGetCellIsReadOnly | Event called to determine if a cell is editable. |

| | |
|---|---|
| OnGetCellTextColor | Event called to determine the cell text color. |
| OnGetColumnHeader | Event called to get the header of a column. |
| OnGetColumnMaximumWidth | Event called to get the maximum width of a column. |
| OnGetColumnMinimumWidth | Event called to get the minimum width of a column. |
| OnGetColumnWidth | Event called to get the width of a column. |
| OnGetNumberOfColumns | Event called to get the number of columns in the TableView. |
| OnGetNumberOfItems | Event called to get the number of items (rows) in the TableView. |
| OnGetSizeToFitWidthOfColumn | Event called to get the size to fit the width of a column. |
| OnGetValueForItem | Event called to get the value for a specific item. |
| OnIsColumnResizable | Event called to know if a column is resizable or not. |
| OnIsColumnVisible | Event called to know if a column is visible or not. |
| OnSelectionDidChange | Event called if the selection did change. |
| OnSelectionIsChanging | Event called if the selection is changing. |
| OnSetValueForItem | Event called if the value for an item is set. |

**Adding Columns and Items**

The TableView has a columns collection and an items collection. The items collection has Values collection that can be used to add values according to the number of columns. An item on a specific column is called a cell.

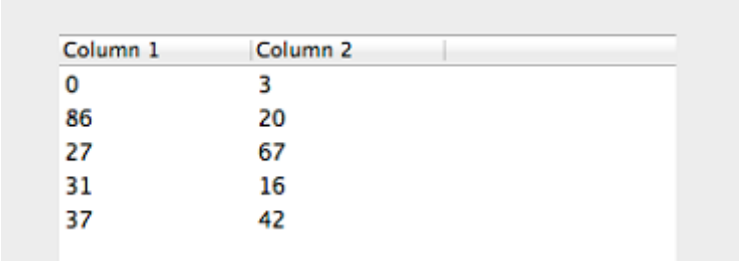
A Column can be added with the following code:

```
var
  c: TTMSFMXNativeNSTableViewColumn;
begin
  TMSFMXNativeNSTableView1.BeginUpdate;
  c := TMSFMXNativeNSTableView1.Columns.Add;
  c.Header := 'Column 1';
```

```
  c := TMSFMXNativeNSTableView1.Columns.Add;
  c.Header := 'Column 2';
  TMSFMXNativeNSTableView1.EndUpdate;
end;
```


To add values to each column, you need to access the items collection which has a values collection that is mapped on the columns. The code below adds 2 columns and 5 items for each column.

```
var
  c: TTMSFMXNativeNSTableViewColumn;
  I: Integer;
  R: Integer;
  it: TTMSFMXNativeNSTableViewItem;
begin
  TMSFMXNativeNSTableView1.BeginUpdate;
  c := TMSFMXNativeNSTableView1.Columns.Add;
  c.Header := 'Column 1';
  c := TMSFMXNativeNSTableView1.Columns.Add;
  c.Header := 'Column 2';

  for R := 0 to 4 do
  begin
    it := TMSFMXNativeNSTableView1.Items.Add;
    for I := 0 to TMSFMXNativeNSTableView1.Columns.Count - 1 do
    begin
      it.Values.Add.Value := Random(100);
    end;
  end;

  TMSFMXNativeNSTableView1.EndUpdate;
end;
```

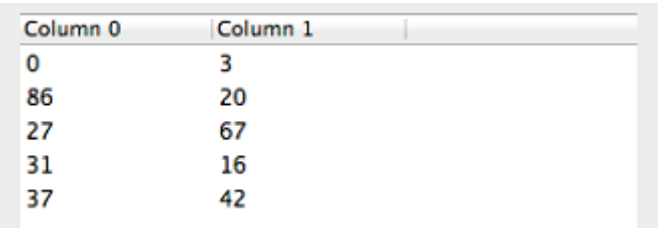


The Value property can hold different types of data: string, double, integer, boolean and TBitmap values. When adding Boolean values and TBitmap values, the boolean values are automatically converted to a Checkbox and the TBitmap values to an image.

**Virtual mode**

The TableView also supports a virtual mode where you can specify how many columns and items need to be displayed as well as the value for that item. The code below implements four events to achieve a virtual mode, with item values, and column headers. The code has a similar output as the code in the previous sample. To make a difference in the output, the Columns header text start from index 0 instead of 1.

```
procedure TForm1.TMSFMXNativeNSTableView1GetColumnHeader(Sender: TObject;
  AColumn: Integer; var AHeader: string);
begin
  AHeader := 'Column ' + inttostr(AColumn);
end;

procedure TForm1.TMSFMXNativeNSTableView1GetNumberOfColumns(Sender:
TObject;
  var ANumberOfColumns: Integer);
begin
  ANumberOfColumns := 2;
end;

procedure TForm1.TMSFMXNativeNSTableView1GetNumberOfItems(Sender: TObject;
  var ANumberOfItems: Integer);
begin
  ANumberOfItems := 5;
end;

procedure TForm1.TMSFMXNativeNSTableView1GetValueForItem(Sender: TObject;
  AColumn, AItem: Integer; var AValueForItem: TValue);
begin
  AValueForItem := Random(100);
end;
```

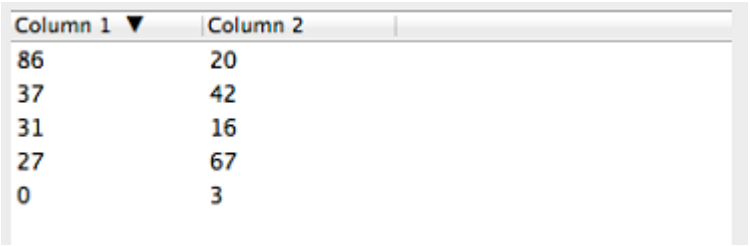



**Sorting**

Sorting is enabled by default, and displays a sort indicator when clicking on a column. Only a single column can be sorted, by clicking on the header of the column. Sorting can only be executed in non-virtual mode. Taking our first sample, clicking on the first column will automatically sort the data ascending:

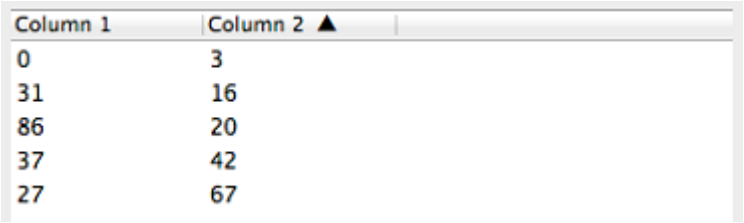
| Column 1 ▲ | Column 2 |
|---|---|
| 0 | 3 |
| 27 | 67 |
| 31 | 16 |
| 37 | 42 |
| 86 | 20 |

Clicking the column again, will automatically sort the first column in the other direction (descending):



| Column 1 ▼ | Column 2 |
|---|---|
| 86 | 20 |
| 37 | 42 |
| 31 | 16 |
| 27 | 67 |
| 0 | 3 |

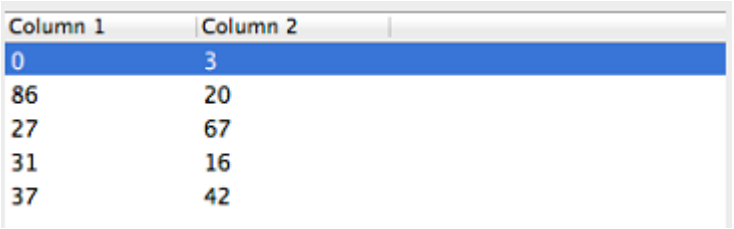
Clicking on the second column will sort the data based on the second column, the indicator is removed, and the column is sorted.



| Column 1 | Column 2 ▲ |
|---|---|
| 0 | 3 |
| 31 | 16 |
| 86 | 20 |
| 37 | 42 |
| 27 | 67 |

**Selection**

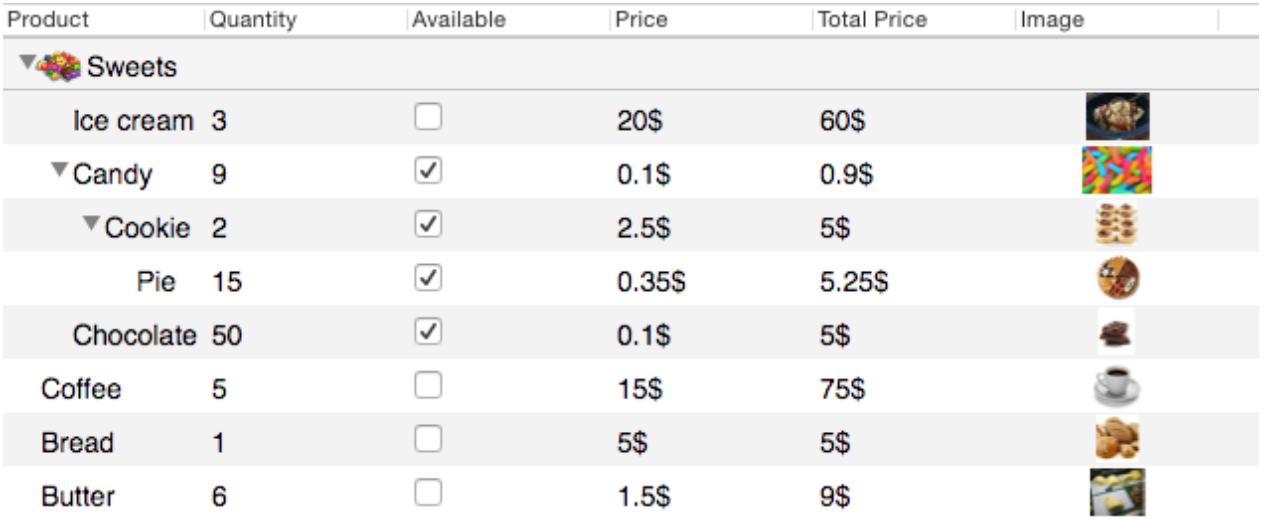
The TableView does not require selection by default. By setting the Options.Interaction.EmptySelection to false, the first row is automatically selected. If EmptySelection is false, the TableView selection can be cancelled by clicking in an empty area of the TableView.



| Column 1 | Column 2 |
|---|---|
| 0 | 3 |
| 86 | 20 |
| 27 | 67 |
| 31 | 16 |
| 37 | 42 |

The selected item / row can be retrieved by calling TMSFMXNativeNSTableView.SelectedRow and if Options.Interaction.MultipleSelection is set to true, the TMSFMXNativeNSTableView.SelectedRows will return an array of integers.

## TMSFMXNativeNSOutlineView




**Usage**

A TMSFMXNativeNSOutlineView displays data in a columns and rows structure.

**Published Properties**

| | |
|---|---|
| Columns | A collection of columns. |
| Columns[Index] → Alignment | The text alignment for all the cells in a column. |
| Columns[Index] → CellColor | The color for all cells in a column. |
| Columns[Index] → CellTextColor | The text color for all cells in a column. |
| Columns[Index] → Header | The header of a column. |
| Columns[Index] → MaximumWidth | The maximum width of a column. |
| Columns[Index] → MinimumWidth | The minimum width of a column. |
| Columns[Index] → ReadOnly | Enables / disables editing in a column. |
| Columns[Index] → Reordering | Enables / disables reordering of a column. |
| Columns[Index] → ResizingMask | The resizing mask of a column. |

| | |
|---|---|
| Columns[Index] → Visible | Sets a column visibility. |
| Columns[Index] → Width | The width of a column. |
| Items | A collection of items (rows). |
| Items[Index] → Values | A collection of row values for each column. |
| Items[Index] → GroupItem | Specifies if an item is a group item and applies a different style when true. |
| Items[Index] → Items | A collection of sub items. |
| Values[Index] → Value | The value of a cell on a specific row and column. The type of this property is TValue and currently supports String, Double, Integer, Boolean and TBitmap. |
| Values[Index] → Icon | The icon of a cell on a specific row and column. |
| Options | A set of options to configure the OutlineView. |
| Options → Appearance | The appearance of the OutlineView. |
| Options → Appearance → AlternatingRowBackgroundColors | Enables / disables the alternating row background colors for the OutlineView. |
| Options → Appearance → BackgroundColor | The background color of the OutlineView. |
| Options → Appearance → GridColor | The grid color of the OutlineView. |
| Options → Appearance → SelectionHighlightStyle | The selection style for the OutlineView. |
| Options → Interaction | The interaction of the OutlineView. |
| Options → Interaction → ColumnReordering | Allows column reordering on the OutlineView. |
| Options → Interaction → ColumnResizing | Allows column resizing on the OutlineView. |
| Options → Interaction → ColumnSelection | Allows column selection on the OutlineView. |
| Options → Interaction → EmptySelection | Allows an empty selection in the OutlineView. |
| Options → Interaction → MultipleSelection | Allows selecting multiple rows in the OutlineView. |
| Options → Layout | The layout of the OutlineView. |

| | |
|---|---|
| Options → Layout → ColumnAutoResizingStyle | Sets the column auto resizing style for the OutlineView. |
| Options → Layout → ColumnHeaders | Enables / disables the column header of the OutlineView. |
| Options → Layout → ColumnSpacing | Spacing between columns. |
| Options → Layout → RowHeight | The row height of the OutlineView. |
| Options → Layout → RowSizeStyle | The style of the row size of the OutlineView. |
| Options → Layout → RowSpacing | The spacing between rows of the OutlineView. |

**Public Properties**

| | |
|---|---|
| OutlineView | Returns a reference to the native Mac NSOutlineView. |

**Events**

| | |
|---|---|
| OnAllowColumnReorder | Event called to determine if a column can be reordered or not. |
| OnCellClick | Event called when a cell is clicked. |
| OnCellDblClick | Event called when a cell is double clicked. |
| OnCellEditingCheckBoxClick | Event called when a cell with a checkbox is clicked. |
| OnCellEditingDidBegin | Event called when a cell editing did begin. |
| OnCellEditingDidChange | Event called when a cell editing did change. |
| OnCellEditingDidEnd | Event called when a cell editing did end. |
| OnCellEditingWillChange | Event called when a cell editing will change. |
| OnColumnClick | Event called when a column is clicked. |
| OnColumnDblClick | Event called when a column is double clicked. |
| OnColumnDidDrag | Event called when a column is dragged. |
| OnColumnDidMove | Event called when a column is moved. |

| | |
|---|---|
| OnColumnDidResize | Event called when a column is resized. |
| OnGetCellAlignment | Event called to determine the cell text alignment. |
| OnGetCellColor | Event called to determine the cell color. |
| OnGetCellIsReadOnly | Event called to determine if a cell is editable. |
| OnGetCellTextColor | Event called to determine the cell text color. |
| OnGetColumnHeader | Event called to get the header of a column. |
| OnGetColumnMaximumWidth | Event called to get the maximum width of a column. |
| OnGetColumnMinimumWidth | Event called to get the minimum width of a column. |
| OnGetColumnWidth | Event called to get the width of a column. |
| OnGetNumberOfColumns | Event called to get the number of columns in the OutlineView. |
| OnGetNumberOfItemsForItem | Event called to get the number of items or subitems in the OutlineView. |
| OnGetSizeToFitWidthOfColumn | Event called to get the size to fit the width of a column. |
| OnGetValueForItem | Event called to get the value for a specific item. |
| OnIsColumnResizable | Event called to know if a column is resizable or not. |
| OnIsColumnVisible | Event called to know if a column is visible or not. |
| OnSelectionDidChange | Event called if the selection did change. |
| OnSelectionIsChanging | Event called if the selection is changing. |
| OnSetValueForItem | Event called if the value for an item is set. |

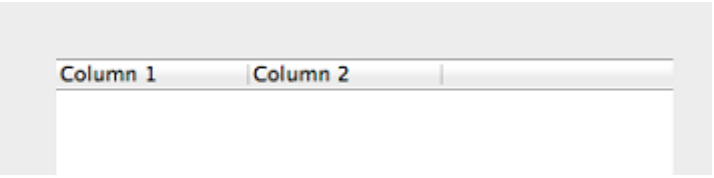
**Adding Columns and Items / Subitems**

The OutlineView has a columns collection and an items / subitems collection. The items collection has Values collection that can be used to add values according to the number of columns. An item

on a specific column is called a cell. Each item can contain multiple sub items and an expand/collapse indicator is automatically shown when the item has sub items.

A Column can be added with the following code:

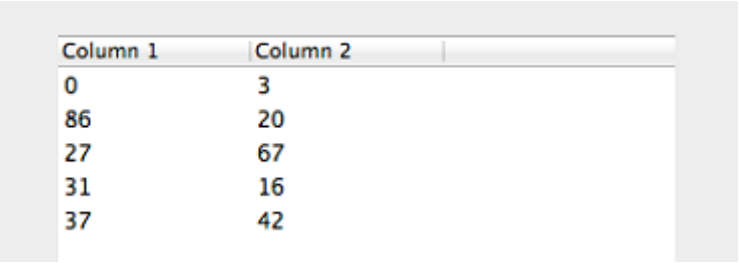
```
var
  c: TTMSFMXNativeNSOutlineViewColumn;
begin
  TMSFMXNativeNSOutlineView1.BeginUpdate;
  c := TMSFMXNativeNSOutlineView1.Columns.Add;
  c.Header := 'Column 1';
  c := TMSFMXNativeNSOutlineView1.Columns.Add;
  c.Header := 'Column 2';
  TMSFMXNativeNSOutlineView1.EndUpdate;
end;
```




To add values to each column, you need to access the items collection which has a values collection that is mapped on the columns. The code below adds 2 columns and 5 items for each column.

```
var
  c: TTMSFMXNativeNSOutlineViewColumn;
  I: Integer;
  R: Integer;
  it: TTMSFMXNativeNSOutlineViewItem;
begin
  TMSFMXNativeNSOutlineView1.BeginUpdate;
  c := TMSFMXNativeNSOutlineView1.Columns.Add;
  c.Header := 'Column 1';
  c := TMSFMXNativeNSOutlineView1.Columns.Add;
  c.Header := 'Column 2';

  for R := 0 to 4 do
  begin
    it := TMSFMXNativeNSOutlineView1.Items.Add;
    for I := 0 to TMSFMXNativeNSOutlineView1.Columns.Count - 1 do
    begin
      it.Values.Add.Value := Random(100);
    end;
  end;

  TMSFMXNativeNSOutlineView1.EndUpdate;
```

```
end;
```

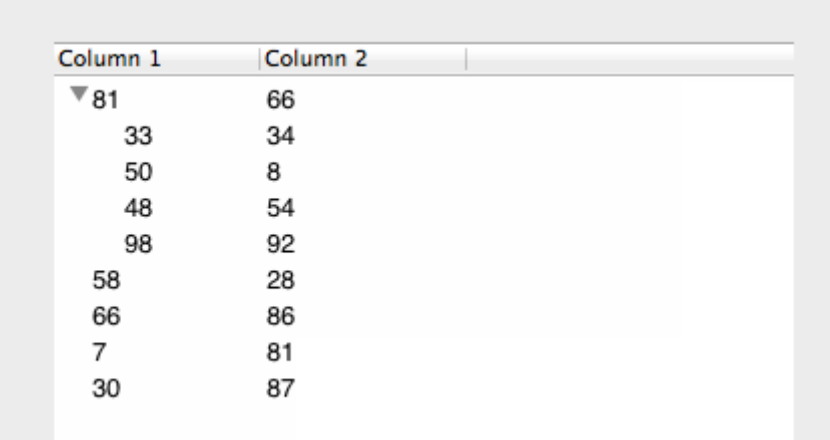



The Value property can hold different types of data: string, double, integer, boolean and TBitmap values. When adding Boolean values and TBitmap values, the boolean values are automatically converted to a Checkbox and the TBitmap values to an image.

Adding subitems is done by accessing the items collection for a specific item. In the above sample, the first item is used to add 5 additional subitems. When expanding the first item, the added sub items are shown.

```
var
  c: TTMSFMXNativeNSOutlineViewColumn;
  I: Integer;
  R: Integer;
  it, itSub: TTMSFMXNativeNSOutlineViewItem;
  K: Integer;
begin
  TMSFMXNativeNSOutlineView1.BeginUpdate;
  c := TMSFMXNativeNSOutlineView1.Columns.Add;
  c.Header := 'Column 1';
  c := TMSFMXNativeNSOutlineView1.Columns.Add;
  c.Header := 'Column 2';

  for R := 0 to 4 do
  begin
    it := TMSFMXNativeNSOutlineView1.Items.Add;
    for I := 0 to TMSFMXNativeNSOutlineView1.Columns.Count - 1 do
      it.Values.Add.Value := Random(100);

    if R = 0 then
    begin
      for K := 0 to 3 do
      begin
        itSub := it.Items.Add;
```

```
        for I := 0 to TMSFMXNativeNSOutlineView1.Columns.Count - 1
do
          itSub.Values.Add.Value := Random(100);
      end;
    end;
  end;

  TMSFMXNativeNSOutlineView1.EndUpdate;
```




**Virtual mode**

The OutlineView also supports a virtual mode where you can specify how many columns and items/sub items need to be displayed as well as the value for that item/sub item. The code below implements four events to achieve a virtual mode, with item values, and column headers. The code has a similar output as the code in the previous sample. To make a difference in the output, the Columns header text starts from index 0 instead of 1.

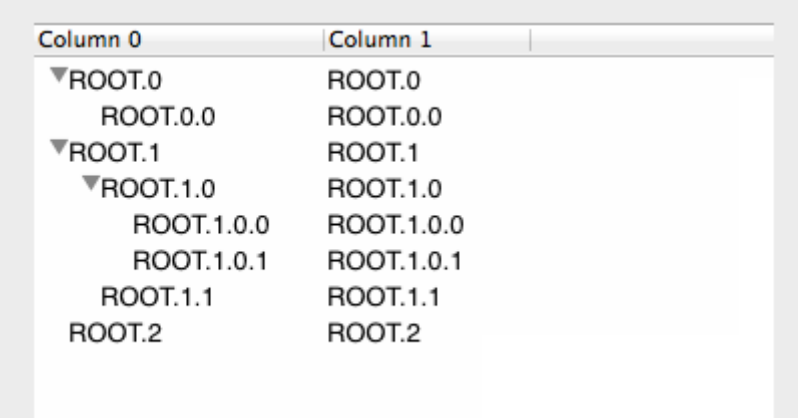
Note that the items/sub items structure is based on a node string that starts with 'ROOT', which is the first level of items. Each level that follows is suffixed with the index of the item. A typical node layout in virtual mode can be structured as the sample below.

Each event / method that needs to access its node data / structure and the column/row information uses the TTMSFMXNativeNSOutlineViewNodeItem record.

```
ROOT
        ROOT.0
                ROOT.0.0
        ROOT.1
                ROOT.1.0
                        ROOT.1.0.0
```

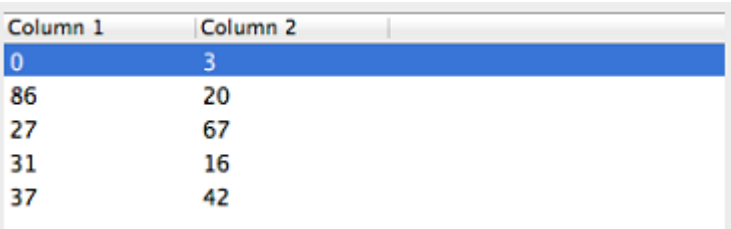
ROOT.1.0.1
ROOT.1.1
ROOT.2

```
procedure TForm1.TMSFMXNativeNSOutlineView1GetColumnHeader(Sender:
TObject;
  AColumn: Integer; var AHeader: string);
begin
  AHeader := 'Column ' + inttostr(AColumn);
end;

procedure
TForm1.TMSFMXNativeNSOutlineView1GetNumberOfColumns(Sender: TObject;
  var ANumberOfColumns: Integer);
begin
  ANumberOfColumns := 2;
end;

procedure TForm1.TMSFMXNativeNSOutlineView1GetNumberOfItemsForItem(
  Sender: TObject; AItem: TTMSFMXNativeNSOutlineViewNodeItem;
  var ANumberOfItems: Integer);
begin
  if AItem.ANode = 'ROOT' then
  begin
    ANumberOfItems := 3;
  end
  else if AItem.ANode = 'ROOT.0' then
  begin
    ANumberOfItems := 1;
  end
  else if AItem.ANode = 'ROOT.1' then
    ANumberOfItems := 2
  else if AItem.ANode = 'ROOT.1.0' then
    ANumberOfItems := 2;
end;

procedure TForm1.TMSFMXNativeNSOutlineView1GetValueForItem(Sender:
TObject;
  AItem: TTMSFMXNativeNSOutlineViewNodeItem; var AValueForItem:
TValue);
begin
  AValueForItem := AItem.ANode;
end;
```



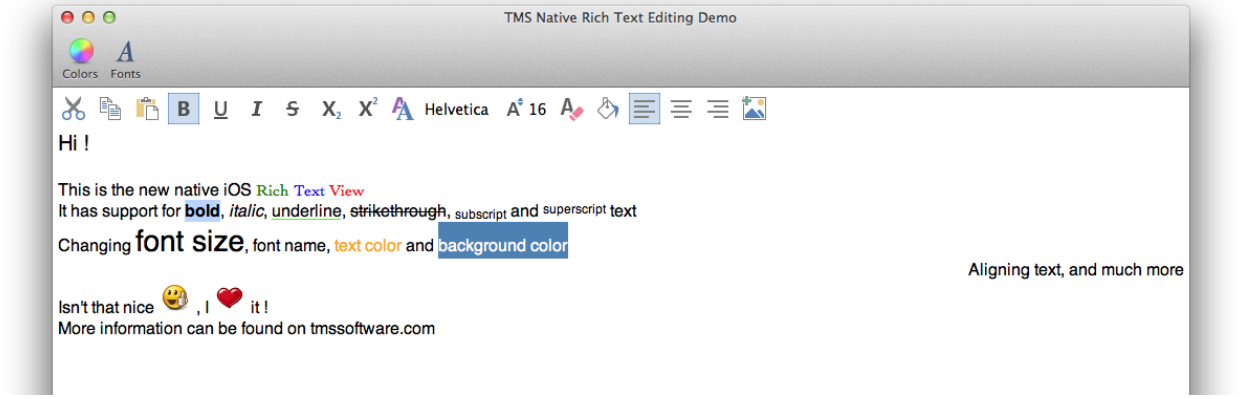

**Selection**

The OutlineView does not require selection by default. By setting the Options.Interaction.EmptySelection to false, the first row is automatically selected. If EmptySelection is false, the TableView selection can be cancelled by clicking in an empty area of the OutlineView.




The selected item / row can be retrieved by calling TMSFMXNativeNSOutlineView.SelectedRow and if Options.Interaction.MultipleSelection is set to true, the TMSFMXNativeNSOutlineView.SelectedRows will return an array of integers. To retrieve a specific item / subitem under the selected row you can use the TMSFMXNativeNSOutlineView.SelectedItem instead.

# TMSFMXNativeNSRichTextView




**Usage**

This component is based on the native NSTextView component and adds rich text editing capabilities. For more information about properties, methods and events that are not listed here please refer to the TMSFMXNativeNSTextView component.

**Public Properties**

| | |
|---|---|
| Selection: TTMSFMXNativeNSRichTextViewRange | 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 | Gets and Sets a compatible Archived XML String that can be used to persist the rich text contents of the TextView. |

**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

TMSFMXNativeNSRichTextView1.SetBold(True);


//apply bold to the text at position 5 with a length of 3

TMSFMXNativeNSRichTextView1.SetBold(True, 5, 3);
```

| | |
|---|---|
| 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 ALineHeight parameter > -1. Also, by default, the bitmap is inserted at selection, unless the ALocation 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 ALineHeight parameter > -1. Also, by default, the bitmap is inserted at selection, unless the ALocation 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 | 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. |
| 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 GetUnderline / SetUnderline functionality. |
| GetStrikethroughColor / SetStrikethroughColor | 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 | Gets or Sets the text subscript value offset. Can be combined with SetFontSize for a smaller font. |
| GetSuperscript / SetSuperscript | Gets or Sets the text superscript value offset. Can be combined with SetFontSize for a smaller font. |
| GetTextLength | Returns the length of the text of a TextView. |
| GetTooltip / SetTooltip | Gets or Sets a tooltip on the text. When hovering with the mouse over the text, the toolip will be shown. |
| 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 | Gets or Sets the text underline color. |
| GetURL / SetURL | Gets or Sets the text URL. The URL is only clickable when the TextView Editable 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. |
| ImportFromStream / ExportToStream | Functionality to import / export the rich text from / to a memory stream. |

| | |
|---|---|
| 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. |
| 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). |

**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, HTML, Microsoft Word Document, Open Office XML,… formatted data.

**Only RTFD files can be used to import / export data with image support.**

# TMSFMXNativeNSRichTextViewToolbar


**Usage**

This component can be used separately or in combination with a TMSFMXNativeNSRichTextView 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 customizing this and hiding specific actions. Set under Options the correct value to false to hide a specific action.
When the TMSFMXNativeNSRichTextViewToolBar is connected to a TMSFMXNativeNSRichTextView (via assigning a TMSFMXNativeNSRichTextView to the TMSFMXNativeNSRichTextViewToolBar.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 TMSFMXNativeNSRichTextView.

When no TMSFMXNativeNSRichTextView is connected to the TMSFMXNativeNSRichTextViewToolBar, the triggered event for the action can be used to programmatically apply the appropriate attribute.

# TMSFMXNativeMaciCloud

**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-ValueDataIniCloud.html

**Methods**

| | |
|---|---|
| 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 AutoSynchronize 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 UpdateKeys |
| UpdateKeys | Forces a synchronize operation and retrieves all |

| | |
|---|---|
| | keys from the iCloud key value store. |

**Properties**

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

**Events**

| | |
|---|---|
| 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. |

**Supported types**

The TMSFMXNativeMaciCloud 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.

**Entitlements**

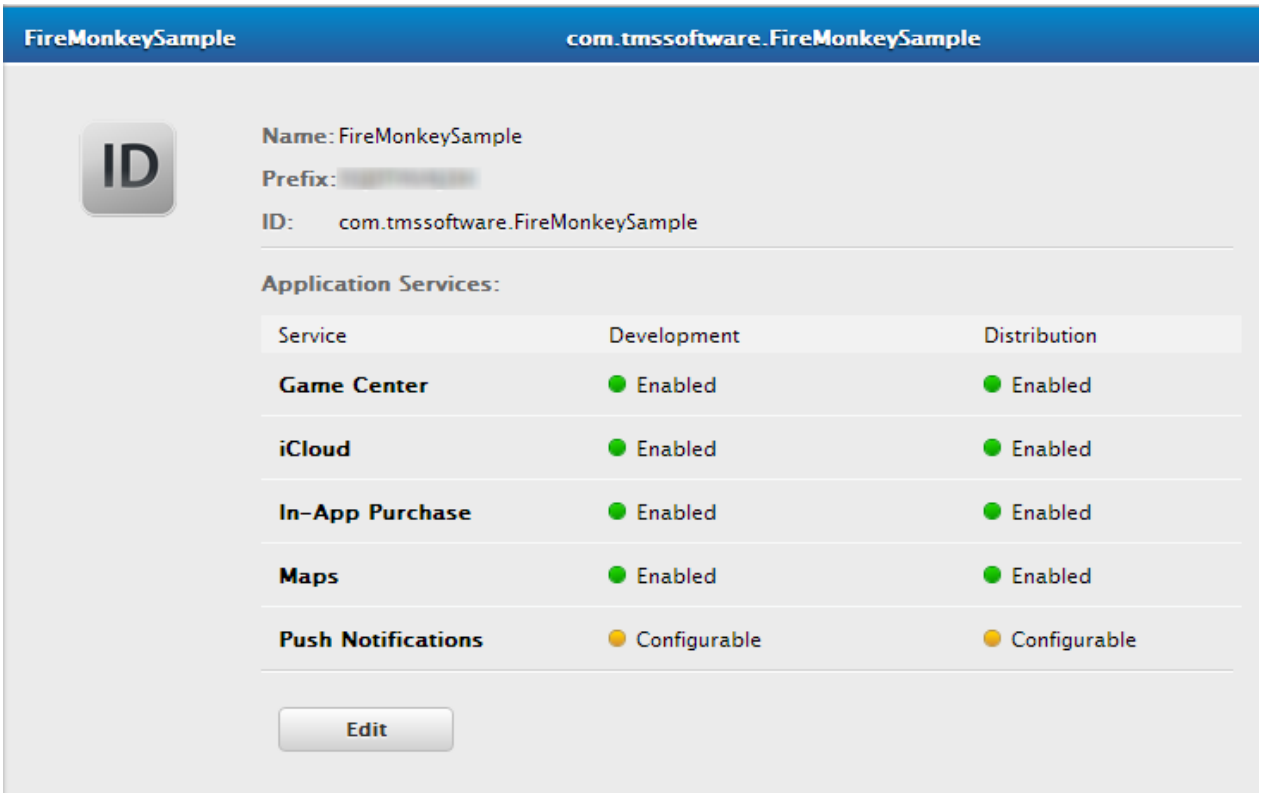
Before iCloud can be used in your application you need to enable it and sign your application. Additional information about enabling iCloud and incorporating it into your application can be found on the following page:

https://developer.apple.com/library/Mac/documentation/General/Conceptual/iCloudDesignGuide/Chapters/iCloudFundametals.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. 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>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 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, used to generate a provisioning profile to sign your application.




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**.

The signing itself cannot be done through RAD Studio in the same way as the signing is done for the iOS iCloud variant. The signing needs to be done manually after compiling / deploying your sample. More information about signing can be found on the following page:

http://docwiki.embarcadero.com/RADStudio/XE5/en/Mac_OS_X_Application_Development

A good signing tool that allows you to specify the entitlements for your application is App Wrapper:

http://www.ohanaware.com/appwrapper/

# TMSFMXNativeMaciCloudDocument

**Usage**

The TMSFMXNativeMaciCloudDocument 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

**Properties**

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

**Methods**

| | |
|---|---|
| AddDocument | Adds a new document to the Documents collection and moves the file to iCloud. When the file is added, the OnDocumentAdded 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 LoadDocuments is the first step you need to manually implement after iCloud has been initialized. The OnInitialized 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 | Manually refresh the documents asynchronously. When the documents are refreshed, the OnDocumentsRefreshed event is called. This event is also called when there are changes in the iCloud document storage.<br><br>Each refresh automatically calls OnDocumentAdded, OnDocumentDeleted and OnDocumentUpdated based on the difference of the current and the previous documents state. The Documents collection is automatically updated. |
| RemoveDocument | Removes the document from the collection and moves an existing document from iCloud to a local directory. |
| SwitchContainer | Switches between containers, after the ContainerIdentifier has been set. The currently loaded documents are cleared and renewed with the documents in the other container. If the ContainerIdentifier 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. |

**Events**

| | |
|---|---|
| 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 LoadDocuments. |
| 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. |

**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 `TMSFMXNativeMaciCloudDocument1.SwitchContainer;`

```
procedure TForm1.TMSFMXNativeMaciCloudDocument1Initialized(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.TMSFMXNativeMaciCloudDocument1DocumentsLoaded(Sender:
TObject);
var
  doc: TTMSFMXNativeMaciCloudDocumentItem;
  I: Integer;
begin
  ListBox1.BeginUpdate;
  ListBox1.Clear;
  for I := 0 to TMSFMXNativeMaciCloudDocument1.DocumentCount - 1 do
  begin
    doc := TMSFMXNativeMaciCloudDocument1.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.

**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.

**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:

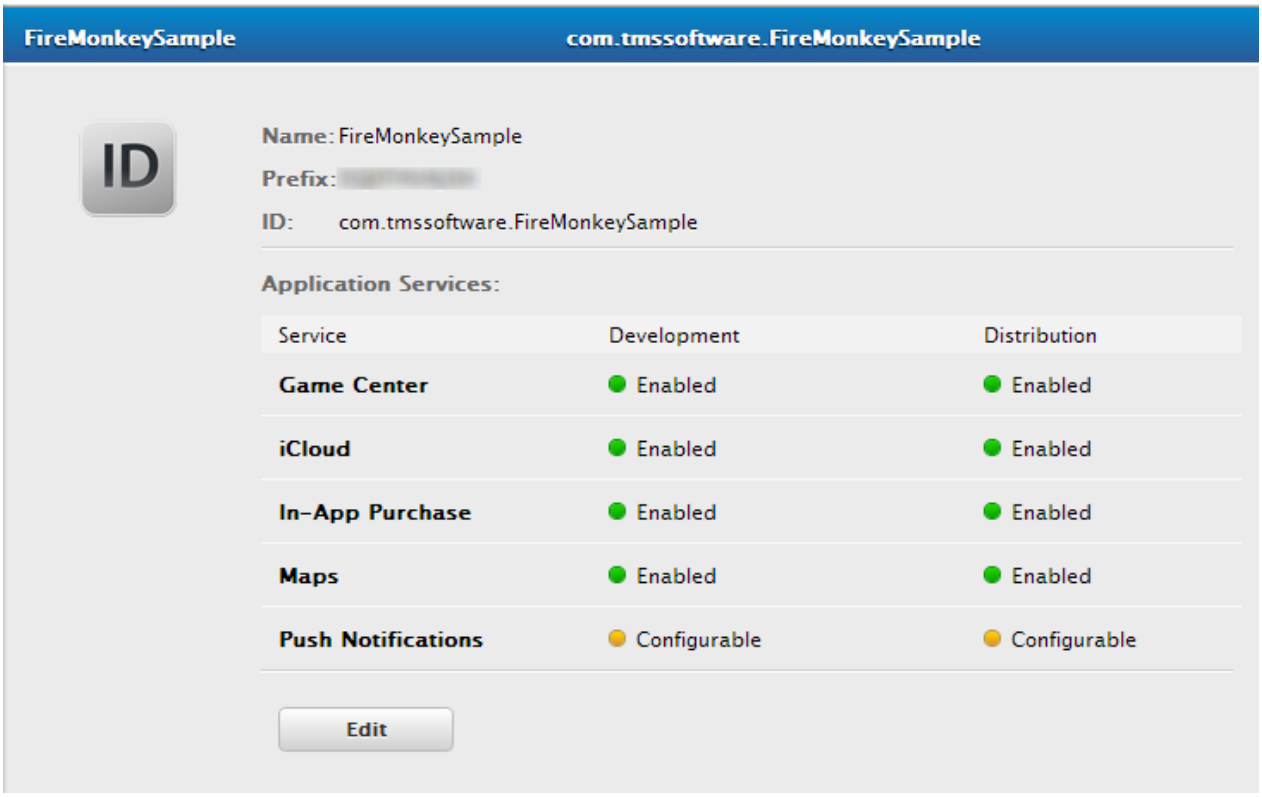
https://developer.apple.com/library/Mac/documentation/General/Conceptual/iCloudDesignGuide/Chapters/iCloudFundametals.html

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. 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>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 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, used to generate a provisioning profile to sign your application.




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**.

The signing itself cannot be done through RAD Studio in the same way as the signing is done for the iOS iCloud variant. The signing needs to be done manually after compiling / deploying your sample. More information about signing can be found on the following page:

http://docwiki.embarcadero.com/RADStudio/XE5/en/Mac_OS_X_Application_Development

A good signing tool that allows you to specify the entitlements for your application is App Wrapper:

http://www.ohanaware.com/appwrapper/

# TMSFMXNativeMacPDFLib

**Usage**

The TMSFMXNativeMacPDFLib 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.

**Methods**

| | |
|---|---|
| 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 TMemoryStream instance when calling EndDocument. |
| 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 memorystream, depending on the chosen action in BeginDocument. |
| 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 GetPageInfo method returns the various boxes (MediaBox, CropBox, TrimBox, ArtBox and BleedBox) 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 | Boolean to determine if a PDF document was already opened with OpenDocument. |
| 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 TMemoryStream |
| SaveDocumentFromStream(FileStream: TMemoryStream; FileName: String); | Saves an existing PDF document from a TMemoryStream to a file. |
| UnlockWithPassword(Password: String): Boolean; | When an existing PDF document is opened, the contents might be encrypted when calling GetDocumentInfo. This method unlocks the document with a password and returns a Boolean if the document is unlocked successfully. |

**Public Properties**

| | |
|---|---|
| MediaBox, TrimBox, BleedBox, ArtBox, CropBox | Various boxes that can be used to retrieve / set information from / to a page or document that is created / opened.<br><br>The MediaBox is used to specify the width and height of the page. The MediaBox is the largest page box in a PDF. The other page boxes can equal the size of the MediaBox but they cannot be larger.<br><br>The CropBox defines the region to which the page contents are to be clipped.<br><br>The BleedBox determines the region to which the page contents needs to be clipped when output in a production environment.<br><br>The TrimBox defines the intended dimensions of the finished page. Contrary to the CropBox, the TrimBox is very important because it defines the actual page size.<br><br>The ArtBox is a bit of a special case. It can define a region within a page that is of special interest. |
| ModificationDate | The date the PDF document was modified. |
| CreationDate | The date the PDF document was created. |

| | |
|---|---|
| Producer | The producer of the PDF document. |

**Properties**

| | |
|---|---|
| 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 GetPageInfo, 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 GetPageInfo, 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. |
| 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 UnlockWithPassword method. |

**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.

```
TMSFMXNativeMacPDFLib1.BeginDocument('FileName');
TMSFMXNativeMacPDFLib1.NewPage;
TMSFMXNativeMacPDFLib1.EndDocument;
```

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

```
var
  ms: TMemoryStream;
begin
  TMSFMXNativeMacPDFLib1.BeginDocument;
  TMSFMXNativeMacPDFLib1.NewPage;
  ms := TMSFMXNativeMacPDFLib1.EndDocument;
end;
```

**Opening an existing document**

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

```
TMSFMXNativeMacPDFLib1.OpenDocument('FileName');
if TMSFMXNativeMacPDFLib1.UnlockWithPassword('Password') then
//optional password unlocking
begin
  TMSFMXNativeMacPDFLib1.GetDocumentInfo; // get document
information //such as the Author, Title, …
  TMSFMXNativeMacPDFLib1.GetPageInfo(1); // get page informaton such
as //the MediaBox, CropBox, …
end;
TMSFMXNativeMacPDFLib1.CloseDocument;
```

Opening a document from a memory stream is based on the same code but with a different OpenDocument overload.

**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;
```

**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.

```
TMSFMXNativeMacPDFLib1.BeginDocument('FileName');
TMSFMXNativeMacPDFLib1.NewPage;
TMSFMXNativeMacPDFLib1.FillColor := TAlphaColorRec.Red;
TMSFMXNativeMacPDFLib1.StrokeColor := TAlphaColorRec.Darkred;
TMSFMXNativeMacPDFLib1.LineWidth := 3;
TMSFMXNativeMacPDFLib1.DrawPathBegin;
TMSFMXNativeMacPDFLib1.DrawPathMoveToPoint(PointF(200, 200));
TMSFMXNativeMacPDFLib1.DrawPathAddCurveToPoint(PointF(250, 150),
PointF(325, 250), PointF(200, 300));
TMSFMXNativeMacPDFLib1.DrawPathAddCurveToPoint(PointF(75, 250),
PointF(150, 150), PointF(200, 200));
TMSFMXNativeMacPDFLib1.DrawPathClose;
TMSFMXNativeMacPDFLib1.DrawPathEnd;
TMSFMXNativeMacPDFLib1.EndDocument;
```

**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.

```
TMSFMXNativeMacPDFLib1.BeginDocument('FileName');
TMSFMXNativeMacPDFLib1.NewPage;
TMSFMXNativeMacPDFLib1.RichText.Text := 'Hello World';
TMSFMXNativeMacPDFLib1.RichText.SetBold;
TMSFMXNativeMacPDFLib1.RichText.SetForegroundColor(TAlphaColorRec.Red, 0, 5);
TMSFMXNativeMacPDFLib1.DrawRichText(RectF(50, 50, 150, 100));
TMSFMXNativeMacPDFLib1.EndDocument;
```

**Text Flow**

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 := TMSFMXNativeMacPDFLib1.MediaBox;
InflateRect(r, -50, -50);
TMSFMXNativeMacPDFLib1.BeginDocument('FileName');
TMSFMXNativeMacPDFLib1.NewPage;
TMSFMXNativeMacPDFLib1.DrawText(lorem, RectF(r.Left, R.Top, r.Right, R.Top + 250), 3);
TMSFMXNativeMacPDFLib1.EndDocument;
```

**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.

**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.

# View hierarchy

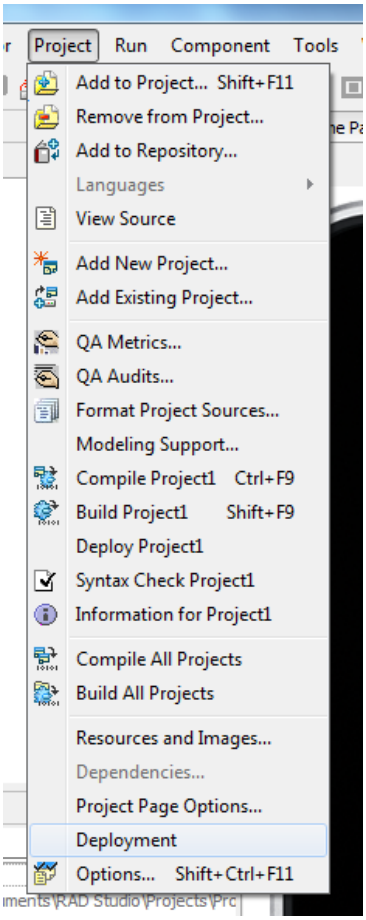
The TMS mCL components can be dropped directly as a child of the main application form, but can also be used as a child of another TMS FMX Native NS control. Included in the set is a TMSFMXNativeNSView control that can be compared with a TPanel in VCL. The view is typically used as a container control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

Drop a TMSFMXNativeNSView on the form and add a TMSFMXNativeNSButton control as child of the view.

When setting the visible property of the TMSFMXNativeNSView to false, the button will also disappear. If we have a large area of controls and need to apply scrolling, the TMSFMXNativeNSScrollView can be used as a container for other controls.

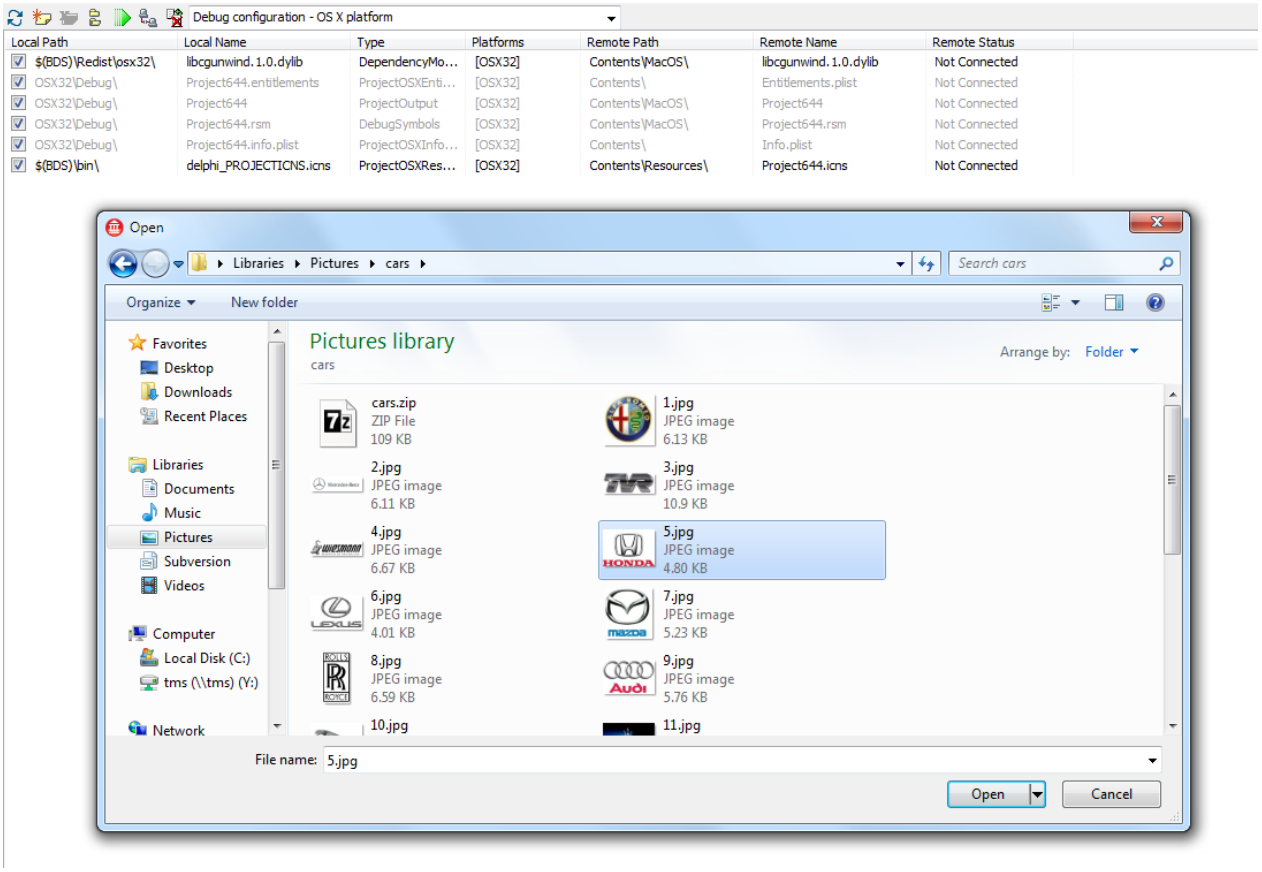
# Deployment

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 application, clicking on the project tab and selecting deployment shows a window where these files can be added.



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

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




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

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

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

Root Directory :

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

```
end;
```