

TMS Pack for FireMonkey
DEVELOPERS GUIDE

April 2016
Copyright © 2016 by tmssoftware.com bvba
Web: http://www.tmssoftware.com
Email: info@tmssoftware.com

Index

Availability 7
Description 7
Windows support 7
List of available controls 9
Shapes 9
Components 9
TMSFMXNavBar 24
TTMSFMXEdit / TTMSFMXEditBtn 25
TTMSFMXIPEdit 27
TTMSFMXLabelEdit 27
TTMSFMXGraphicCheckLabel 28
TTMSFMXPageSlider 29
Properties / Events 30
TTMSFMXTableView 31
Architecture 31
Styling 33
Properties / Methods / Events 34
Item storage and buffering 39
Adding and removing items 40
Sorting 41
Categories 42
Lookup 45
Filtering / Searching 46
Editing / Deleting 47
DetailView 48
Layout 50
User interface interaction with the TableView 53
MultiSelect 53
Additional Item Elements 54
Performance 55
Binding Controls 55
LiveBindings 57
HTML support 60

TTMSFMXPopup 62
TTMSFMXCircularGauge 64
TTMSFMXLinearGauge 68
TTMSFMXJogMeter 69
TTMSFMX7SegLED 70
TTMSFMXCompass 71
TTMSFMXClock 71
TTMSFMXRotarySwitch / TTMSFMXKnobSwitch 72
TTMSFMXMatrixLabel 73
TTMSFMXScope 75
TTMSFMXSpinner 77
TTMSFMXLEDMeter / TTMSFMXLEDScope 79
TTMSFMXLED / TTMSFMXLEDBar 80
TTMSFMXSlider 80
TTMSFMXTileList 81
- Architecture 81
- Styling 82
- Properties / Methods / Events 83
- Adding and removing tiles 86
- Badges 87
- Tile Styles 88
- Columns and Rows 91
- Paging / Scrolling 92
- PageSize 93
- ColumnWidth / RowHeight 94
- Filtering / Searching / Lookup 95
- Keyboard navigation 95
- Reordering tiles 95
- Performance 95
- MultiSelect 96
- LiveBindings 97

TTMSFMXHotSpotImage 97
- Adding a new hotspot 98
- Magic Wand Tool 100
- Saving and loading hotspots 101

Compatibility 101
TTMSFMXHotSpotEditorDialog 101
TTMSFMXSpeedButton 102
TTMSFMXCalendar / TTMSFMXCalendarPicker 102
TTMSFMXTrackBar 106
TTMSFMXButton 106
TTMSFMXColorSelector / TTMSFMXColorPicker 107
TTMSFMXBitmapSelector / TTMSFMXBitmapPicker 107
TTMSFMXFontNamePicker / TTMSFMXFontSizePicker 109
TTMSFMXWebBrowser / TTMSFMXWebBrowserPopup 110
TTMSFMXSignatureCapture 112
TTMSFMXListEditor 112
Architecture 112
Appearance 112
Items 113
Events 113
TTMSFMXTaskDialog 114
TTMSFMXCheckGroup / TTMSFMXRadioGroup TTMSFMXCheckGroupPicker / TTMSFMXRadioGroupPicker 116
TTMSFMXToolBar 118
Set of components 118
Adding new components at designtime 119
Adding new components at runtime 120
Toolbar button 120
TTMSFMXDateTimeEdit 122
TTMSFMXRatingGrid 122
TTMSFMXPassLock 124
TTMSFMXTouchKeyboard / TTMSFMXPopupTouchKeyboard 127
Description 127
Properties & Events 127
Methods 128
TTMSFMXScrollMenu 129
Description 129
Properties & Events 129
Methods 130

TTMSFMXGridFilterPanel 131
TTMSFMXRichEditor 137
Description 137
Organization 137
Getting Started 138
Properties & Events 138
Methods 139
Programmatic access to the document 144
Using merge fields 146
Using accompanying toolbars 148
Importing & exporting in rich text 148
Importing & exporting in HTML format 149
Exporting to PDF 149
Import or export to mini-HTML 151
TTMSFMXTouchKeyboard / TTMSFMXPopupTouchKeyboard 152
Description 152
Properties & Events 152
Methods 153
Styling 154
Export to PDF 155
TMS Mini HTML rendering engine 156
Samples 159
TMS TableView Overview Demo 159
TMS TableView LiveBindings Demo 1 & 2 160
TMS Instrumentation WorkShop Demo 166
TMS PageSlider Demo 167
TMS TileList Demo 168
TMS TileList LiveBindings Demo 169
TMS HotSpotImage Demo 170
General FireMonkey component usage guidlines 171
Visual part 171
Non-visual part 171
Naming convention 171

Styling 171

Components 175

Cross-platform deployment of applications with TMS Pack for FireMonkey components 176

Availability

TMS Pack for FireMonkey is a component set that is suitable for cross platform development with the Embarcadero FireMonkey framework and is designed for use with Win32, Win64, Mac OS-X, iOS and Android operating systems. With the registered version of TMS Pack for FireMonkey, FireMonkey HD applications for Windows and Mac OS-X can be created as well as FireMonkey iOS and Android mobile applications that can be deployed to iPhone, iPad, iPod, or Android devices.

Versions:

TMS Pack for FireMonkey requires Delphi XE6 & C++Builder XE6 or newer releases.

Description

TMS Pack for FireMonkey contains components for use in user interfaces designed with the Embarcadero FireMonkey framework. The components have been designed from the ground up based on the core concepts of the FireMonkey framework: made up of styles, fully cross-platform, scalable and compatible with FireMonkey's effects, rotation, and livebindings.

Windows support

For Delphi & C++Builder versions before XE8, Windows support in the TMS Pack for FireMonkey is based on the Delphi Chromium Embedded opensource library and can be installed and compiled by following the steps below.
For XE8 and 10 Seattle, this is not needed and the built-in browser is used.

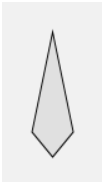
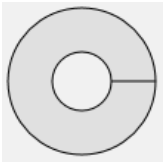
1) Download and Extract http://www.tmssoftware.biz/download/ChromiumFMX.zip
2) Open the package ChromiumFMX.dpk, compile and then install the package.
3) Add the directory where the ChromiumFMX source is located to your Win32 library path in the IDE (ceffmx.pas, ceflib.pas, ...)
4) Navigate to the directory where the TMS Pack for FireMonkey source is installed.
5) Open the file FMX.TMSFMXPackWebBrowser.Win.pas and comment the line {$DEFINE CHROMIUMOFF}
6) Copy the files inside the lib directory from the extracted ChromiumFMX.zip file to the directory where your application executable is or will be located.
7) Create or open a new or existing project and build your project.

Our browser solution for the Windows target is only available in the registered version of TMS Pack for FireMonkey. Windows is NOT supported in the trial version.

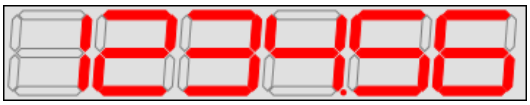
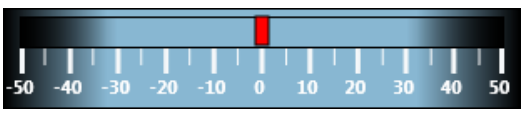
IMPORTANT NOTICE:
If the FireMonkey framework is new to you, please see the chapter “General FireMonkey component usage guidelines” that offers an introduction that is recommended to read before you start working with the TMS Pack for FireMonkey. Another interesting source of information is http://docwiki.embarcadero.com/RADStudio/en/FireMonkey_Application_Platform

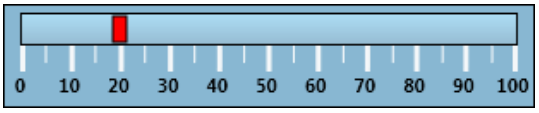
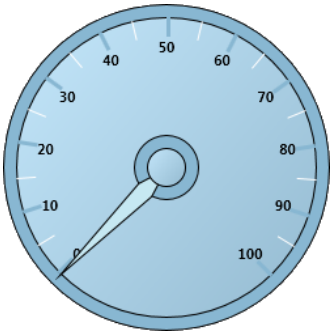
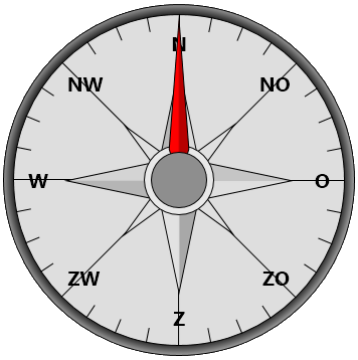
List of available controls

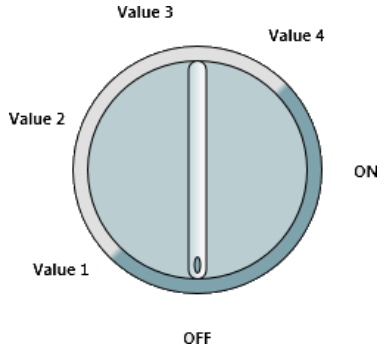
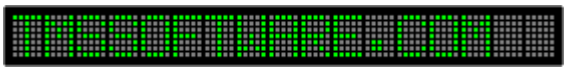
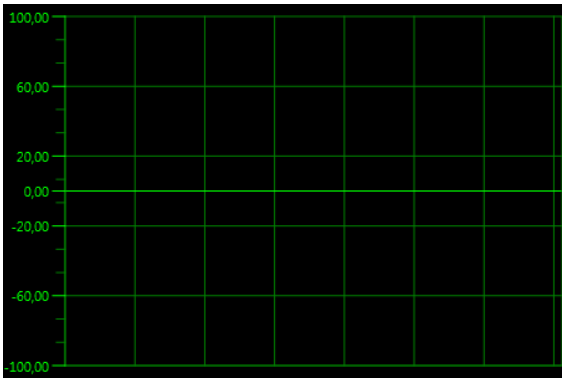
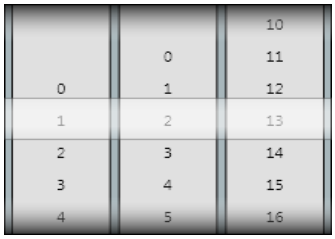
Shapes

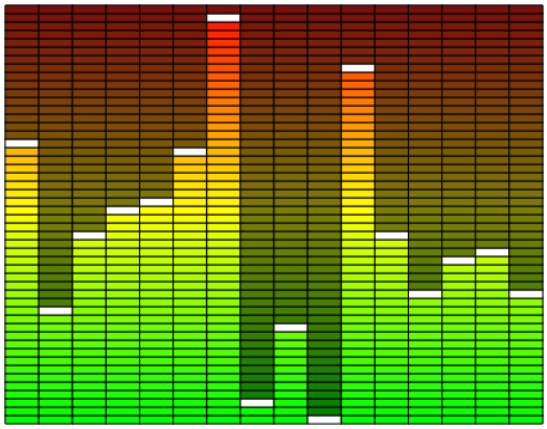
TTMSFMX7SegLEDShape		7-segment LED shape used in the 7-segment LED component
TTMSFMXNeedleShape		A needle shape used in various gauge and meter components.
TTMSFMXSectionShape		A section shape used in the gauge component.
TTMSFMXSetPointShape		A setpoint shape used in the gauge component.
TTMSFMXPieShape		A pie shape used in the circular gauge and variants. This is used to create a 180 ° version for example.

Components

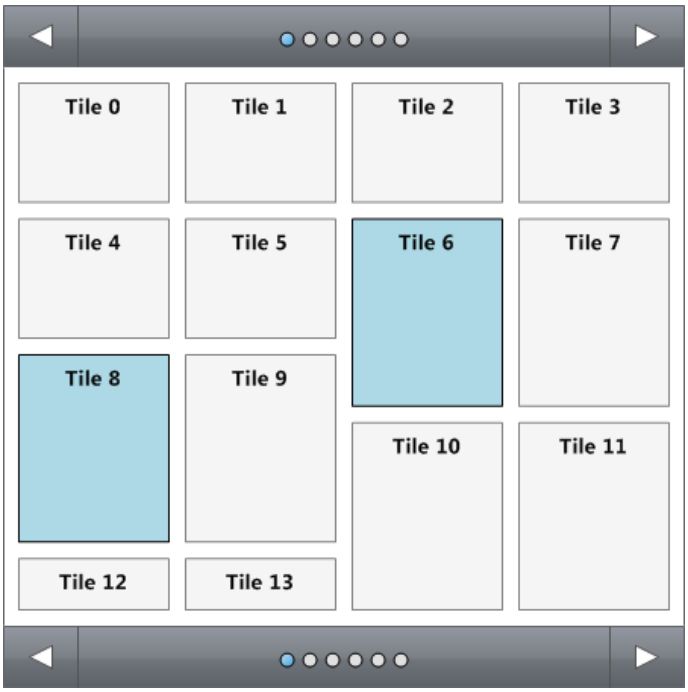
TTMSFMX7SegLED	
TTMSFMXJogMeter	

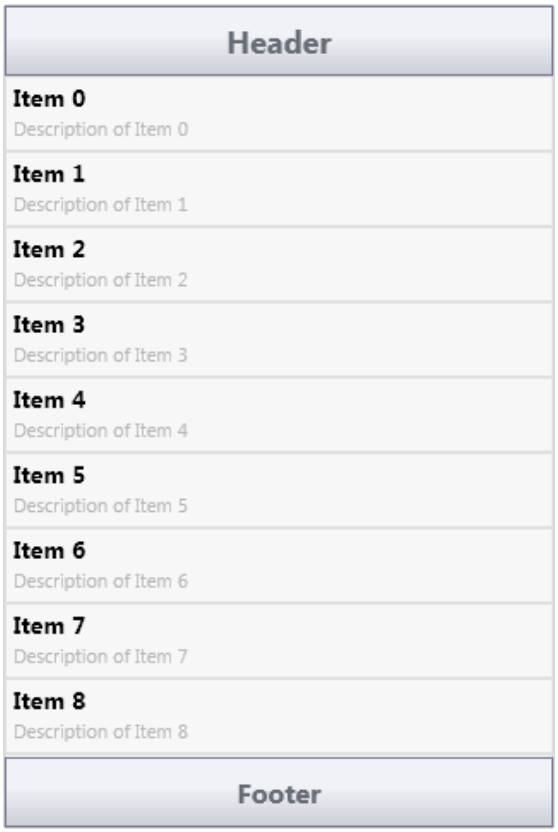
TTMSFMXLinearGauge	
TTMSFMXCircularGauge	
TTMSFMXCompass	
TTMSFMXClock	

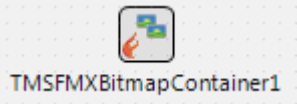
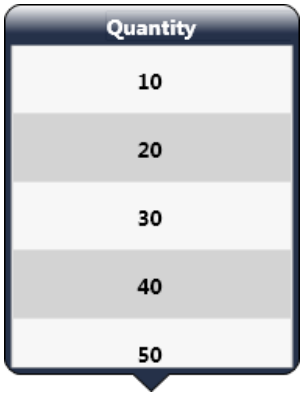
TTMSFMXRotarySwitch	
TTMSFMXKnobSwitch	
TTMSFMXMatrixLabel	
TTMSFMXScope	
TTMSFMXSpinner	

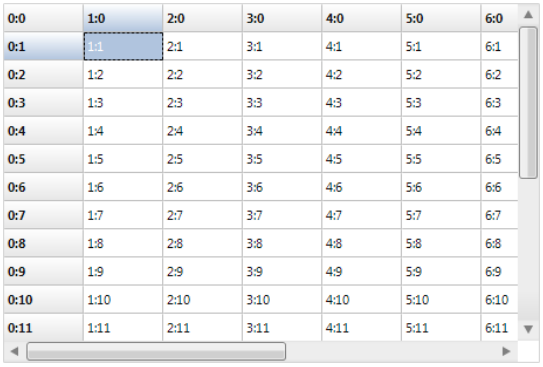
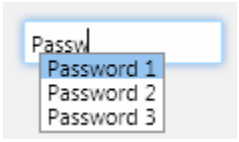
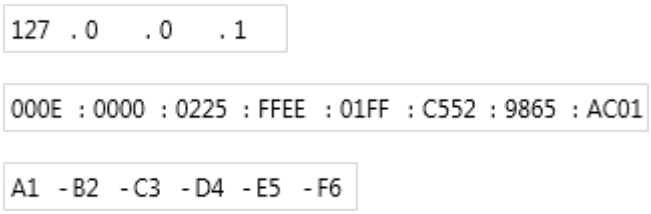
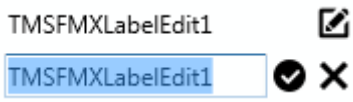
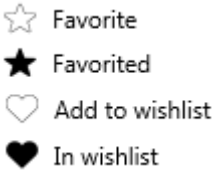
TTMSFMXLEDMeter	
TTMSFMXLEDScope	
TTMSFMXLED, TTMSFMXLEDBar	
TTMSFMXSlider	

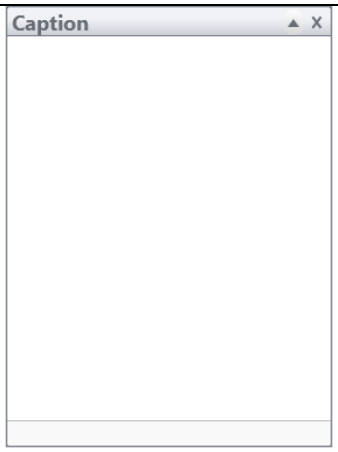
TTMSFMXPageSlider Smoothly animated pager.	

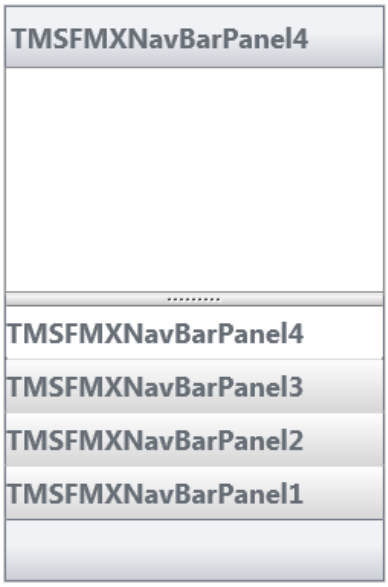
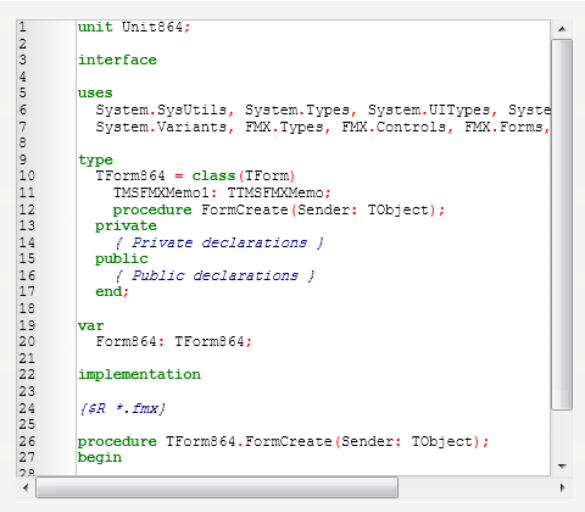
	
TTMSFMXRating	
TTMSFMXTileList	
TTMSFMXBadge	

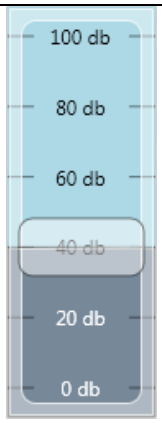
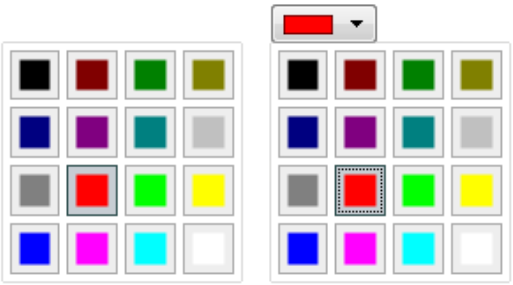
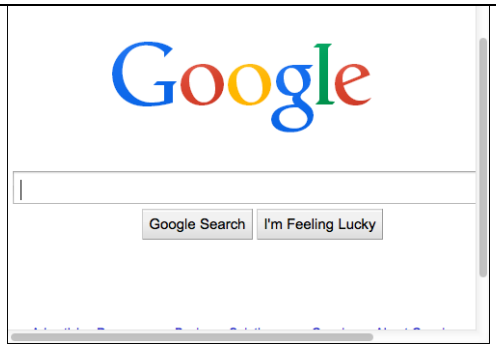
TTMSFMXTableView Smoothly animated scrolling list.	
TTMSFMXHTMLText Text shape that supports HTML (see **MiniHTML chapter**)	
TTMSFMXBitmap Bitmap shape which can load and display an image directly or through a BitmapContainer.	

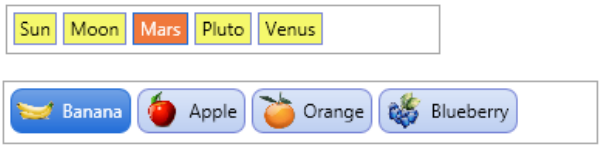
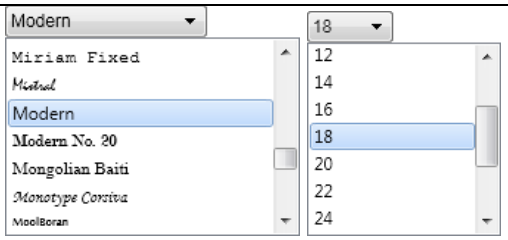
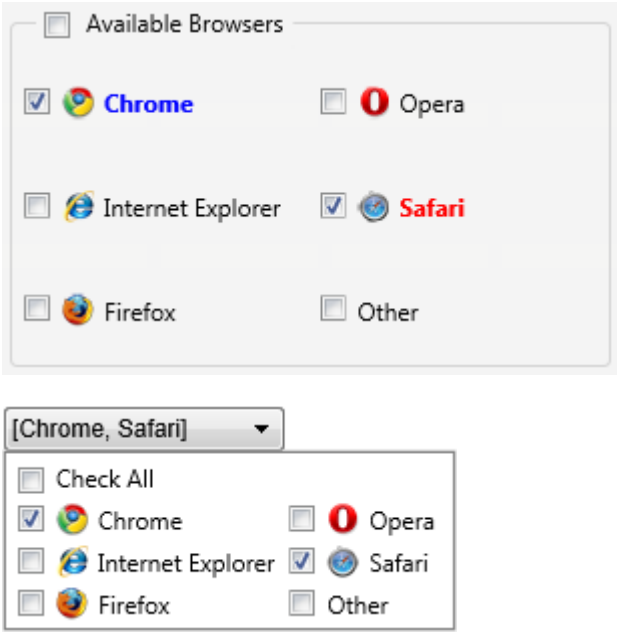
TTMSFMXBitmapContainer Container that holds multiple bitmaps	
TTMSFMXSearchEdit Inherits from TEdit and adds an optional clear and search button, and has a rounded appearance.	
TTMSFMXBarButton Inherits from TButton and adds a Layout and Kind property for different appearances used in the TableView.	
TTMSFMXPopup Component that allows displaying any type of control inside a customizable popup dialog.	
TTMSFMXSpeedButton Inherits from TSpeedButton and adds a Grouping possibility and an image.	
TTMSFMXHotSpotImage Component that is able to display an (optionally stretched) image with predefined clickable and customizable hotspots. Hotspots are added with a complete separate designer.	

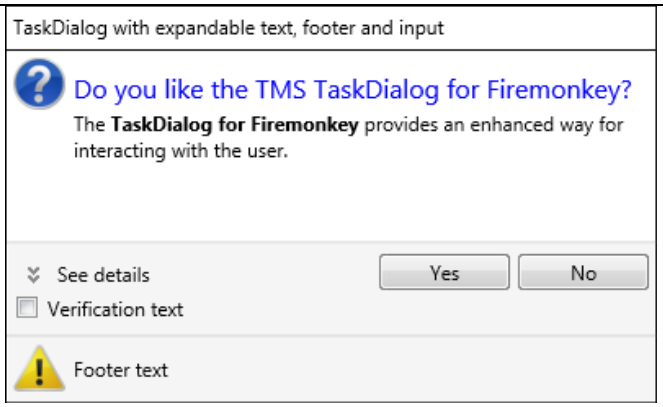
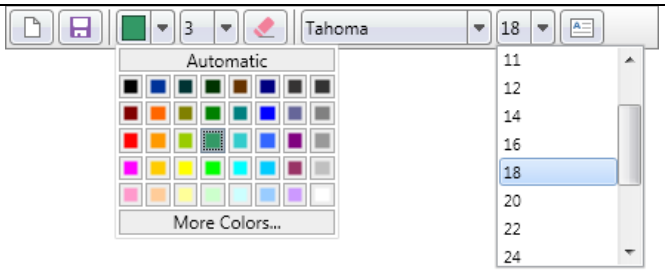
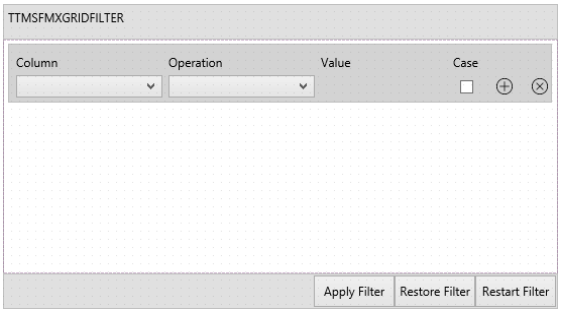
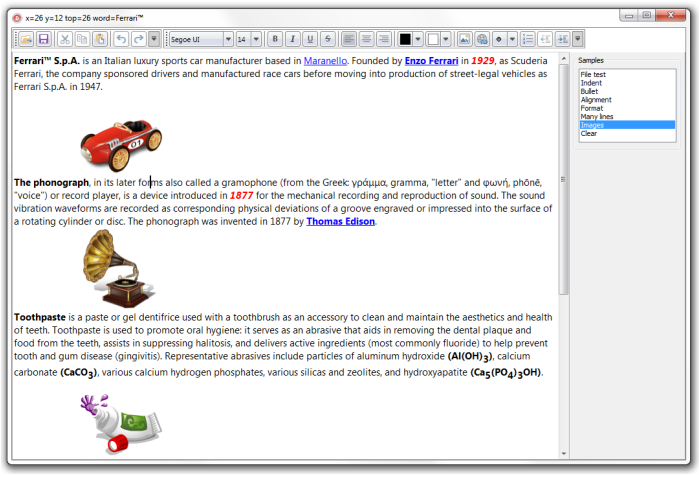
TTMSFMXGrid Feature packed and highly stylable grid component created from scratch. Explained in a separate documentation "TMS Grid for FireMonkey"	
TTMSFMXEdit / TTMSFMXEditBtn Autocomplete and lookup enabled control that extends TEdit. Has the capability of display and editing various editing types such as float, money, lowercase, uppercase, …	
TTMSFMXIPEdit An edit control for inputting IP addresses.	
TTMSFMXLabelEdit A label with a built-in inplace editor.	
TTMSFMXGraphicCheckLabel A checkbox with configurable checkmark image.	
TTMSFMXPanel Expandable / Collapsable container for other controls.	

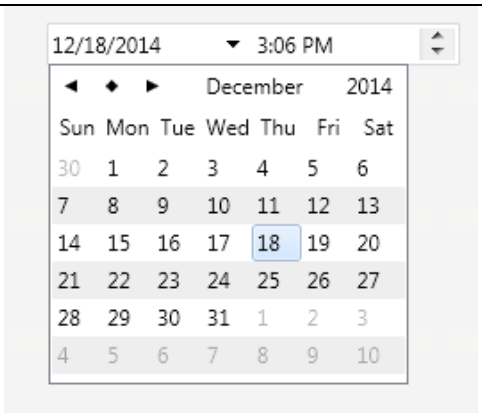
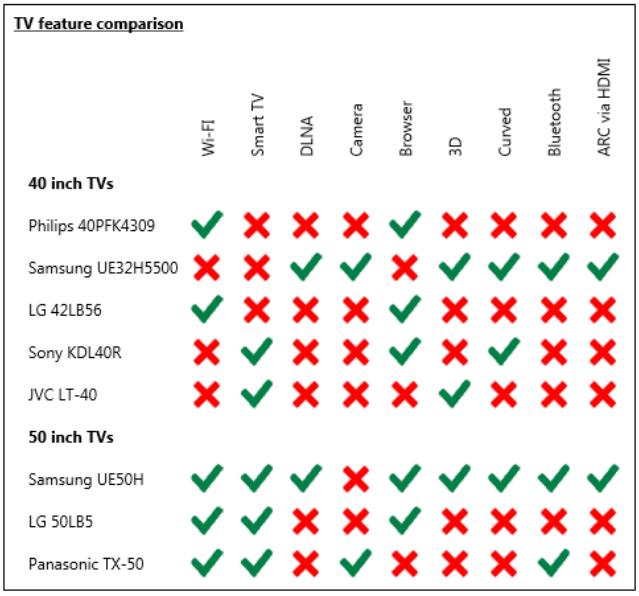
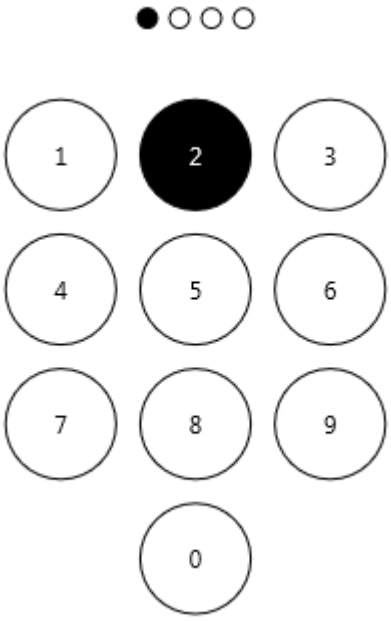
	
TTMSFMXProgressBar Extends TProgressBar and adds a display text with format capabilities.	

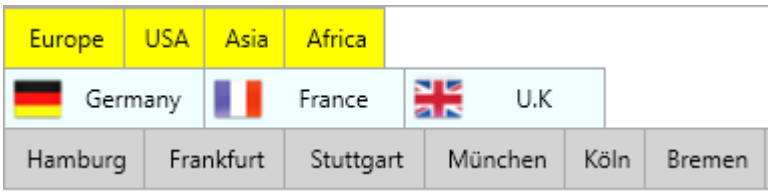
TTMSFMXNavBar Navigation bar with several panels that can hold controls and can be divided in sections.	
TTMSFMXCalendar / TTMSFMXCalendarPicker Calendar with multiselect, disjunctselection, as well as the capability of displaying weeknumbers and weekdays.	
TTMSFMXMemo Feature packed and highly stylable memo component. Explained in a separate documentation "TMS Memo for FireMonkey"	

TTMSFMXTrackBar Highly styleable and configurable trackbar with optional snapping, tickmarks and formatted values text.	
TTMSFMXColorPicker / TTMSFMXColorSelector A color selector and picker with many customization / custom drawing options and events.	
TTMSFMXBitmapPicker / TTMSFMXBitmapSelector A bitmap selector and picker with many customization / custom drawing options and events.	
TTMSFMXButton A standard TButton extended with an optional image and html enabled text.	
TTMSFMXWebBrowser / TTMSFMXWebBrowserPopup	

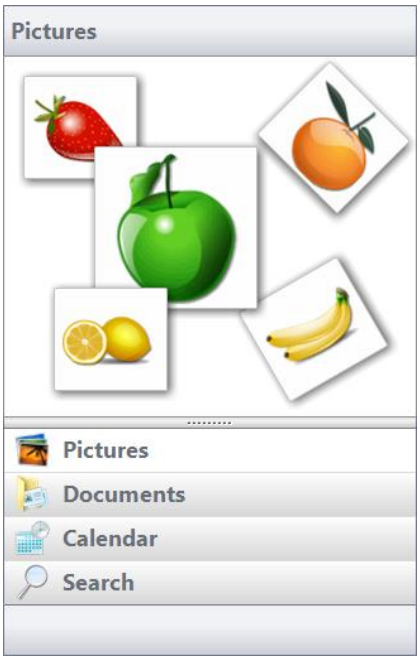
TTMSFMXSignatureCapture	
TTMSFMXListEditor	
TTMSFMXFontNamePicker / TTMSFMXFontSizePicker	
TTMSFMXCheckGroup / TTMSFMXRadioGroup TTMSFMXCheckGroupPicker / TTMSFMXRadioGroupPicker	

TTMSFMXTaskDialog	
TTMSFMXToolBar	
TTMSFMXGridFilterPanel Separate panel for filtering TTMSFMXGrid	
TTMSFMXRichEditor Feature packed and rich editor component	
TTMSFMXDateTimeEdit	

Combination of date and time edit component	
TTMSFMXRatingGrid Control for capturing ratings for different items or for presenting feature comparison lists	
TTMSFMXPassLock Control for capturing passwords on a keypad or circle pattern	

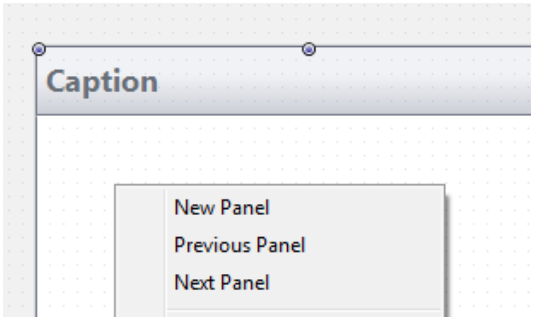
TTMSFMXScrollMenu	
TTMSFMXTouchKeyBoard / TTMSFMXPopupTouchKeyBoard	

TMSFMXNavBar



TTMSFMXNavBar is a container for TTMSFMXNavBarPanel that is able to display several sections per panel. Each panel is displayed in a list that can be collapsed with a slider.

Adding a panel can be done by right-clicking the component and clicking “new panel”.

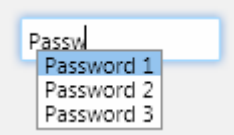


Next panel and previous panel is used to navigate through the panels that you have created.
To create a panel programmatically, first create and then add the panel as a child to the navbar:

```
var
  aPanel: TTMSFMXNavBarPanel;
begin
  aPanel := TTMSFMXNavBarPanel.Create(TMSFMXNavBar1);
  aPanel.Caption := 'Hello World!';
  TMSFMXNavBar1.AddPanel(aPanel);
```

The amount of visible panel items can be controlled with `TMSFMXNavBar1.SplitterPosition` which is limited to the panelcount.

TTMSFMXEdit / TTMSFMXEditBtn



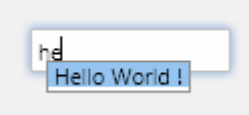
TTMSFMXEdit and TTMSFMXEditBtn extends TEdit and adds several capabilities such as autocompletion, Lookup and supports edit types such as alphanumeric, numeric, float, uppercase, lowercase, money, ….

The lookuplist can be enabled by setting the enabled property to true:

```
TMSFMXEdit1.Lookup.Enabled := True;
```

To display the list while typing, items can be added to the displaylist. The amount of displayed items when typing can be controlled with `TMSFMXEdit1.Lookup.DisplayCount`.

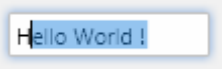
```
TMSFMXEdit1.Lookup.DisplayList.Add('abs');
TMSFMXEdit1.Lookup.DisplayList.Add('Item 1');
TMSFMXEdit1.Lookup.DisplayList.Add('Hello World !');
```



When typing, the list shows after 2 characters, with the property `TMSFMXEdit1.Lookup.NumChars` this can be modified. When typing text, the text that is typed can also be automatically added to the list by setting `TMSFMXEdit1.Lookup.History` to true.

Autocompletion can be actived with `TMSFMXEdit1.AutoComplete := True;` The edit automatically displays the item that matches the characters typed in the edit.

```
TMSFMXEdit1.AutoComplete := True;
TMSFMXEdit1.Lookup.DisplayList.Add('Hello World !');
```

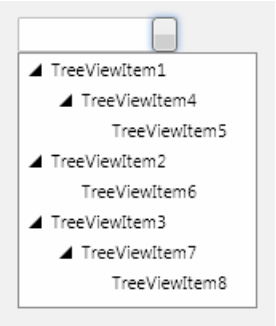


The text in the edit can be displayed as password characters by setting `TMSFMXEdit1.Password := True;`

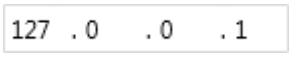
The TTMSFMXEditBtn extends the TTMSFMXEdit, inheriting all features and adds a button to the edit that can display a popupcontrol.

The popupcontrol can be added with

TMSFMXEditBtn1.PopupControl := TreeView1;


TreeViewItem1
TreeViewItem4
TreeViewItem5
TreeViewItem2
TreeViewItem6
TreeViewItem3
TreeViewItem7
TreeViewItem8

TTMSFMXIPEdit



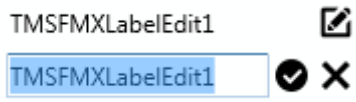




TTMSFMXIPEdit is an edit control for inputting IP addresses. The control supports IPv4, IPv6 and Mac address input as well as automatic masking and focus advance.

Properties
IPAddress: Get or set the IP address
IPAddressType: Set the type of address; IPv4, IPv6 or Mac

TTMSFMXLabelEdit

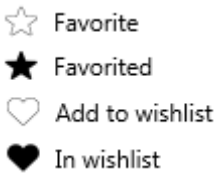


TTMSFMXLabelEdit is a label with a built-in inplace editor. Start editing can be programmatic, by click on Edit button or by click on label. End editing with click on Accept/Cancel button.

Properties
AcceptBitmap / AcceptBitmapName: Assign a custom image for the Accept button
CancelBitmap / CancelBitmapName: Assign a custom image for the Cancel button
EditMaxLength: The maximum number or characters that can be entered
EditMode: Programmatically enable or disable editing
EditType: Define the type of characters that can be entered; Numeric, UpperCase …
EditValidChars: Define the valid characters that can be entered, when EditType is set to etValidChars
EditBitmap / EditBitmapName: Assign a custom image for the Edit button
Text: Get or set the label text

Events
OnEditStart: Event fired when editing is started
OnEditStop: Event fired when editing is ended

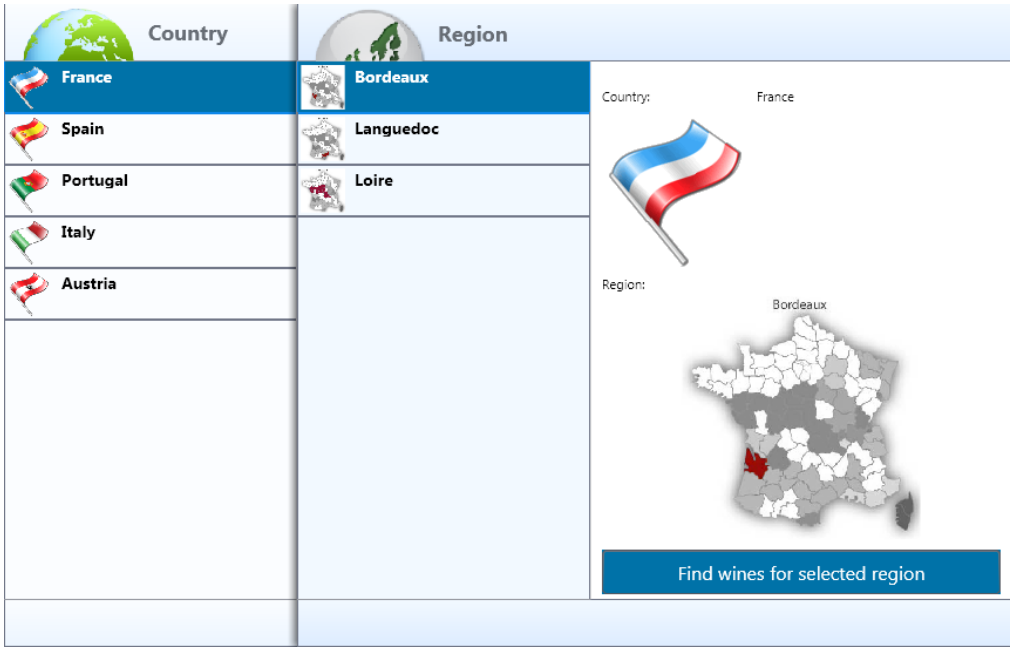
TTMSFMXGraphicCheckLabel

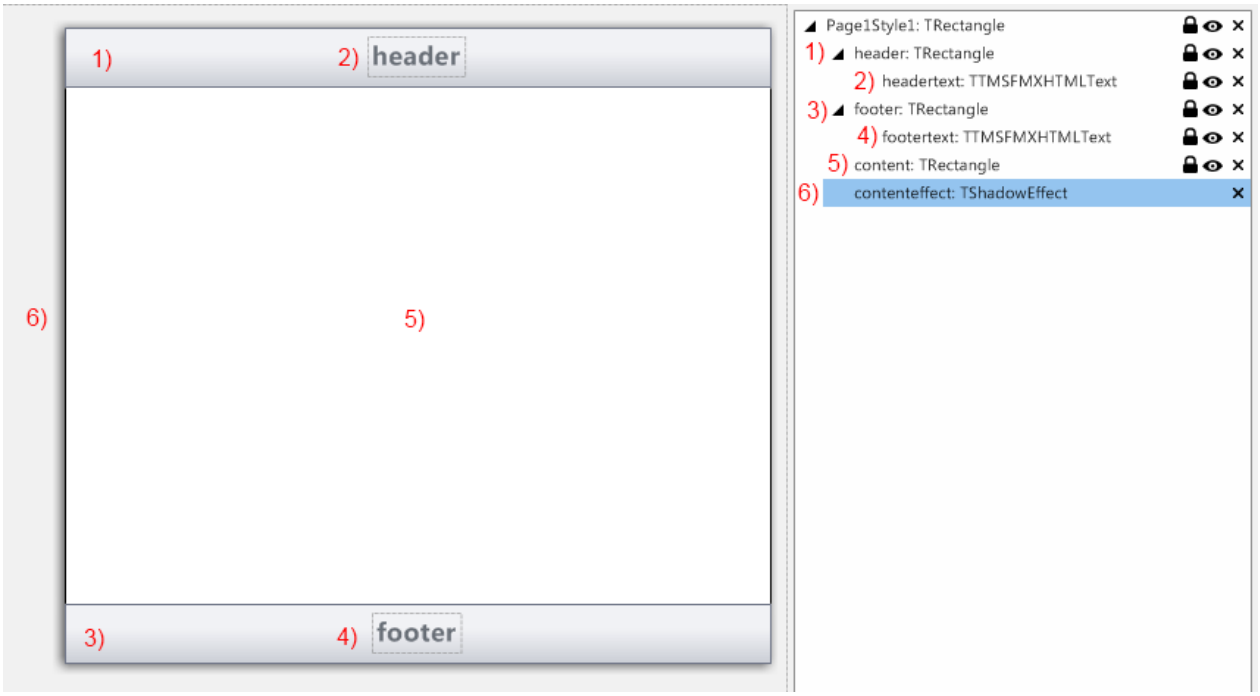


TTMSFMXGraphicCheckLabel is a checkbox with configurable checkmark image. The control has several built-in presets.

Properties
Checked: Programatically change the state to checked / unchecked
Kind: Set the kind of preset images and text to use
StateChecked / StateUnChecked: Configure the checked / unchecked appearance
- **Bitmap / BitmapName**: Assign a custom image
- **Text**: Set the text displayed next to the image

TTMSFMXPageSlider





TTMSFMXPageSlider is a drill down page control that holds and animates multiple pages sliding from left to right and vice versa as drill down is happening.

1) **Header**: The top-aligned rectangle that can hold other controls and contains by default the HTML enabled TTMSFMXHTMLText.
2) **HeaderText**: HTML enabled text.

3) **Footer:** The bottom-aligned rectangle that can hold other controls and contains by default the HTML enabled TTMSFMXHTMLText.
4) **FooterText:** HTML enabled text.
5) **Content:** The client-aligned rectangle that can hold other controls.
6) **Contenteffect**: A shadow effect on the default style

Adding a new page at design time can be done by right-clicking the component and choosing "New page" from the context menu.

To create a new page at runtime the following code can be used:

```
var
  sp: TTMSFMXPage;
begin
  sp:= TTMSFMXPage.Create(TMSFMXPageslider1);
  sp.PageSlider := TMSFMXPageslider1;
  sp.Header := 'Runtime created page';
end;
```

To remove a page at runtime, simply destroy the page:

```
TMSFMXPage.Free;
```

Pages slide in from the right and are animated. To change a page simply set the ActivePageIndex to the correct page. You will notice that pages with a PageIndex that occur before the ActivePageIndex will automatically slide along the Active Page.

Navigation through pages can also be done with the keyboard and the mouse. Click on a page and slide it from right to left to increase the ActivePageIndex or vice versa to decrease. This all happens with animation that can optionally be set faster or slower or be turned off with the AnimationFactor property.

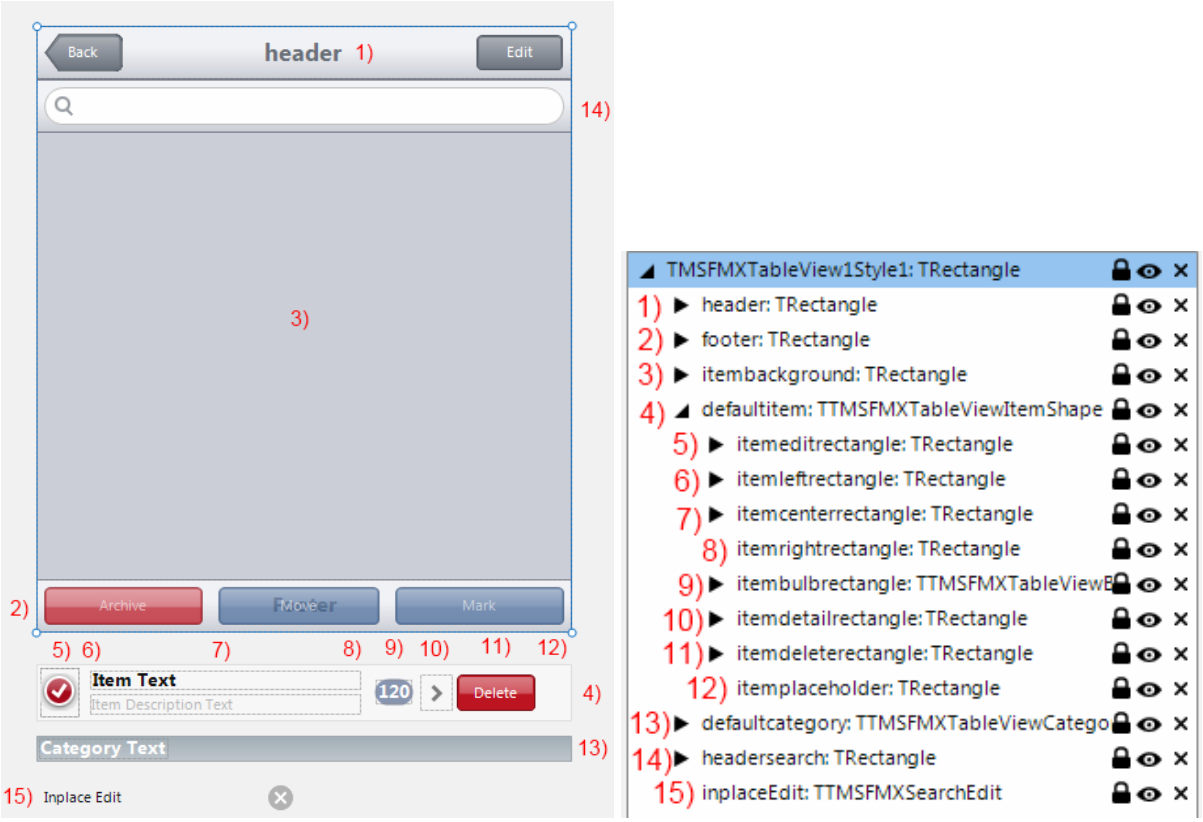
When a page is active, you will notice that the page leaves a minimum amount of width to the left, this is necessary to interact with and show the previous page. This is set with the MinimumWidth property.

Properties / Events

- **ActivePage**: The page that is currently active.
- **ActivePageIndex**: The index of the page that is currently active. This property is used to switch between pages at designtime and runtime.
- **AnimationFactor**: The speed of the animation, the higher the factor the slower the animation.
- **Fill**: The fill of the background of the page.
- **Footer / Header**: The caption of the footer / header which support HTML.
- **MinimumWidth**: The width that is left visible of the page when the page is inactive.
- **PageIndex**: The page index of the page.
- **BitmapContainer**: container of bitmaps used in the Header and Footer to display images with HTML.
- **OnPageChange**: Event called when a page has changed
- **OnPageChanging**: Event called when a page is about to change. With this event page changing can be blocked with the AllowChange parameter.
- **OnPageMoved**: Event called when a page is moved to another PageIndex

TTMSFMXTableView

Architecture



1) **Header**: a rectangle shape with a caption and that can hold optionally a backbutton and an editbutton. The backbutton is used to return from the detail view and the edit button is used to put the tableview in edit mode.

2) **Footer**: a rectangle shape with a caption. When the TableView is in edit mode, the 3 buttons (Archive, Move and Mark) are displayed in this footer rectangle.

3) **BackGround**: the background is a placeholder for the items that also enables scrolling of items and keyboard handling.

4) **DefaultItem**: the basic default layout of the item. When adding items, the layout is copied (cloned) for each item.

5) **Edit Rectangle**: the rectangle that is displayed when the TableView is in edit mode.

6) **Left Rectangle**: a placeholder rectangle that can contain other elements, such as, but not limited to, a checkbox, radiobutton or image.

7) **Center Rectangle**: the rectangle that contains the Caption and Description of the item.

8) **Right Rectangle**: the right rectangle has the same purpose as the left rectangle, but is right aligned.

9) **Bulb Rectangle**: a custom designed shape that can display text in a rounded shaped appearance.

10) **Detail Rectangle**: a rectangle that contains a predefined detail image and is enabled when an item has a DetailView assigned.

11) **Delete Rectangle**: a rectangle that contains a delete button and is enabled when swiping on the item.

12) **PlaceHolder Rectangle**: element internally used to position the elements with extra spacing when the lookupbar / scrollbar are enabled.

13) **Default Category**: the basic layout of a category. When adding items, the category is automatically added according to the kind of item.

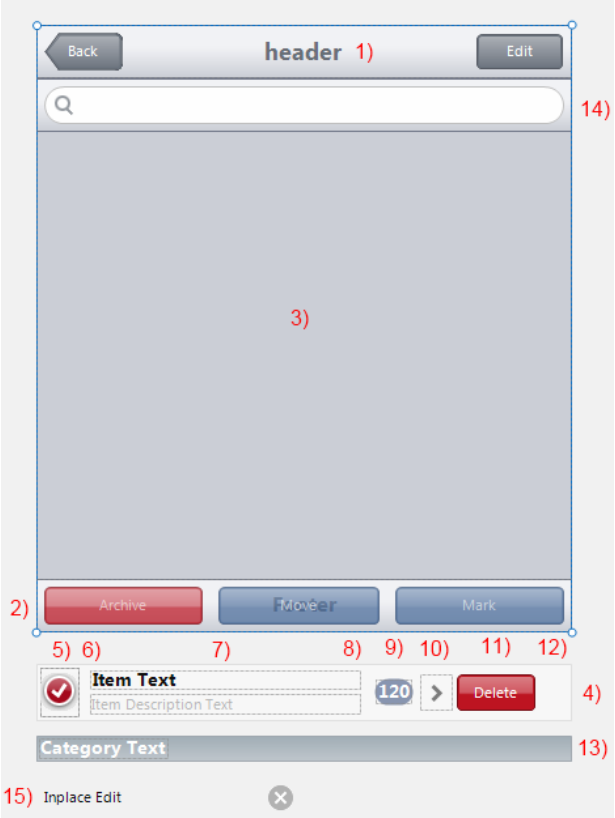
14) **Header Search**: a search edit box displayed when filtering or search is enabled.

15) **Inplace Edit**: search edit box that is transformed to work as an inplaced editor and is displayed when clicking on the caption or description when the appropriate properties are set to enable editing.

Styling

With the FireMonkey design philosophy in mind, we have made the TableView completely styleable. When editing the custom or default style when right-clicking on the component (See: General Firemonkey component usage guidelines) the basic TableView layout is defined with several styleable elements. Elements can be removed, added and modified and updates are reflected in the component when applying the edited style.

Programmatically, almost every element is accessible with a function. When you want to style an element programmatically, you can use the appropriate function. Below is an overview of each element that is styleable at designtime and at runtime:

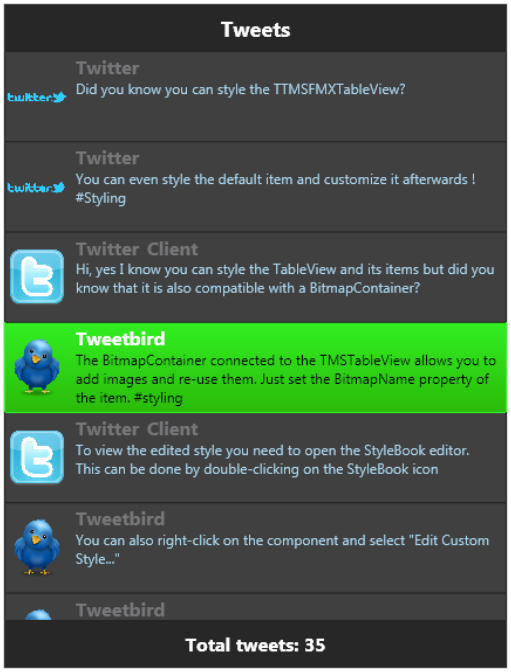


1) `TTMSFMXTableView.GetHeaderRectangle`
2) `TTMSFMXTableView.GetFooterRectangle`
3) `TTMSFMXTableView.GetListBackGround &`
   `TTMSFMXTableView.GetListContainer`
4) `TTMSFMXTableView.GetDefaultItem & TTMSFMXTableViewItem.Shape`
5) `TTMSFMXTableViewItem.ShapeEditRectangle`
6) `TTMSFMXTableViewItem.ShapeLeftRectangle`
7) `TTMSFMXTableViewItem.ShapeCaption &`
   `TTMSFMXTableViewItem.ShapeDescription &`
   `TTMSFMXTableViewItem.ShapeCenterRectangle`
8) `TTMSFMXTableViewItem.ShapeRightRectangle`
9) `TTMSFMXTableViewItem.ShapeBulbRectangle`
10) `TTMSFMXTableViewItem.ShapeDetailRectangle`
11) `TTMSFMXTableViewItem.ShapeDeleteRectangle`
12) `TTMSFMXTableViewItem.ShapePlaceHolder`

13) `TTMSFMXTableView.GetDefaultCategory & TTMSFMXTableViewCategory.Shape`
14) `TTMSFMXTableView.GetSearchEdit`
15) `TTMSFMXTableView.GetInplaceEdit`

These are the most important functions that return the element that is available in the default style or the style of the item when it is cloned. There are more functions available specific for the item and the tableview such as the deletebuttonshape, bulbshape, checked and uncheckedimage and default animations on the backbutton. A complete list can be found when typing the name of the component and search through all the “Get*” functions and for the Tableview Item the functions that have “Shape” in the name.

Below is a sample that shows the power of styling.



Properties / Methods / Events

Below is a list of properties, methods and events in alphabetic order that expose the core functionality of the component and that need a short introduction before delving into the details of the component.

TTMSFMXTableView published properties

- **ArchiveText**: The text of the left button that is displayed in the footer when in edit mode.
- **AutoDeleteItem**: deletes the item automatically.
- **AutoFilter**: When filtering or searching is enabled, the filtering is applied automatically.
- **AutoLoadBuffer**: When buffering is used (BufferSize > 0) the list automatically loads the next buffer when scrolling in the list.
- **AutoLookup**: When using the lookupbar, the list automatically displays the correct category.
- **AutoToggleDetail**: When a DetailView is assigned and the item is clicked. The DetailView is automatically displayed with animation.
- **BackButton**: Displays a back button in the left corner of the header when the DetailView is displayed.

- **BitmapContainer**: Support for BitmapContainer, which is able to contain a collection of reusable TBitmap items.
- **BufferSize**: Used to load an initial buffer of the total collection of items. When scrolling the buffer is repositioned to load the next BufferSize of items.
- **Categories**: Collection of custom categories, used when the CategoryType is set to ctCustom.
- **CategoryType**: The type of categories that are used. A choice can be made between alphabetic, alphanumeric or custom.
- **DefaultDetailView**: The Default detail view that is used by all items if the DetailView property of the item is nil.
- **EditButton**: Displays an edit button in the right corner of the header. This button can be used to set the TableView in edit mode. This button is also used in combination with filtering / searching. In this mode the button is switched to a Cancel button.
- **Filtering**: the type of filtering the list applies. In combination with the search edit box, the vfFilterStart and vfFilterRandom option creates a subset of items and vfSearch highlights the item that matches the search string.
- **FooterText**: The text in the footer.
- **HeaderText**: The text in the header.
- **ItemOptions**: Enables or disables cloning of certain elements that are present in the default item. The performance of the list can be increase when using in combination with the BufferSize property.
- **Items**: a collection of TableView items.
- **LayoutMode**: defines the layout of the item list. The lmNormal option displays the list as a normal TableView and the lmGroup displays the items in grouped mode. The grouped mode used the GroupIndex per item to create groups.
- **LookupBar**: Enables or disables the lookupbar.
- **MarkText**: The text of the right button that is displayed in the footer when in edit mode.
- **MoveText**: The text of the center button that is displayed in the footer when in edit mode.
- **MultiSelect**: Enables or disables multiselect on items.
- **ScrollIndicator**: Shows or hides the scrollindicator that visualizes the amount of items in the collection with an optional fading animation.
- **SelectedItemIndex**: The index of the item that has been selected.
- **ShowFilter**: Show the search edit box on top of the list which can be used to apply filtering or highlight an item in the list.
- **ShowFilterOnSwipe**: Enables or disables the ability of showing the search edit box

Category item published properties in Categories collection

- **Caption**: The Caption displayed in the category.
- **ID**: To link items to the specific category.
- **LookupText**: The text displayed in the lookupbar.

Item published properties in Items collection

- **Bitmap**: The bitmap displayed in the left rectangle image when enabled in the ItemOptions.
- **BitmapName**: The name of the bitmap displayed in the left rectangle image when enabled in the ItemOptions and when a BitmapContainer is assigned and contains an image with the correct name.
- **BulbText**: The text of the bulb rectangle, that is displayed when enabled in the ItemOptions.
- **CanDelete**: When CanDelete is false, the deletebutton will not be displayed and the item cannot be deleted.
- **CanEditCaption**: Displays an inplace editor when clicking on the caption.
- **CanEditDescription**: Displays an inplace editor when clicking on the description.
- **CanSelect**: Enables selection of an item.
- **Caption**: The caption of the item.

- **CategoryID**: The id of the category linked to the item when displaying custom categories.
- **DataBoolean**: Boolean property that is linked to a control placed in the right rectangle. This property also triggers the OnItemData event.
- **DataGroupIndex**: The index that is used in combination with a radiobutton, to create a RadioButtonGroup.
- **DataObject**: Has the same functionality as the DataBoolean property but of type TObject.
- **DataString:** Has the same functionality as the DataBoolean property but of type String.
- **DataValue**: Has the same functionality as the DataBoolean property but of type Single.
- **DeleteButton**: Displays the deletebutton.
- **Description**: The description of the item.
- **DetailView**: The view that is displayed when clicking on an item, to navigate to the detail.
- **GroupIndex**: The index that is used to group items in the lmGroup layoutmode.
- **LeftMargin**: The margin / indenting of an item from the left side.
- **RightMargin**: The margin / indenting of an item from the right side.
- **Selected**: Sets the item selected.
- **Tag**: Tag property that can be used to hold additional information.
- **Visible**: Shows or hides the item.

Public procedures, functions, properties

- ```
  property TopItem: Integer;
  ```
  Sets the first visible item in the list.

- ```
  function SelectedItem: TTMSFMXTableViewItem;
  ```
  Returns the selected item.

- ```
  function CountSelected: Integer;
  ```
  Returns the count of selected items.

- ```
  function ItemFromDifferentCategory(item1, item2:
  TTMSFMXTableViewItem): Boolean;
  ```
  Returns whether the first item is from a different category of the second item.

- ```
  function ItemAtXY(X, Y: Single): TTMSFMXTableViewItem;
  ```
  Returns the Item at a specific position in the TableView.

- ```
  function GetCharacterForItem(AItem: TTMSFMXTableViewItem): String;
  ```
  Returns the Character of the item that is used when categories are applied.

- ```
  function FindItemWithFilter(AFilter: String): TTMSFMXTableViewItem;
  ```
  Returns the item with a specific filter applied.

- ```
  function IsItemInItemList(AItem: TTMSFMXTableViewItem): Boolean;
  ```
  Returns if the item is loaded in the list (depending on the BufferSize and scrollposition)

- ```
  function IsItemVisibleInList(AItem: TTMSFMXTableViewItem): Boolean;
  ```
  Returns if the item is visible in the list.

- ```
  function isEditMode: Boolean;
  ```
  Returns if the edit mode is active or inactive.

- ```
  function isDetailMode: Boolean;
  ```
  Returns if a detail view is visible or not visible.

- ```
  function GetCurrentListView: TControl;
  ```

Returns the current view in edit mode, the main list control or the detail list control.

- `function FindFirstItemWithCategory(ACategory: String): TTMSFMXTableViewItem;`
  Returns the first item with a specific Category when using AlphaBetic / AlphaNumeric categories.

- `function FindFirstItemWithCategoryID(ACategoryID: Integer): TTMSFMXTableViewItem;`
  Returns the first item with a specific CategoryID when using custom categories.

- `procedure ToggleEditMode;`
  Toggles between edit mode and normal mode.

- `procedure ApplyFilter(AFilter: String);`
  Applies and automatically updates the list with a specific filter.

- `procedure UpdateTableView;`
  Forces a complete update of the list of items in the TableView.

- `procedure BeginUpdate;`
  Blocks an update of the TableView. This is necessary when adding a large amount of items. Used in combination with EndUpdate;

- `procedure EndUpdate;`
  Blocks an update of the TableView. This is necessary when adding a large amount of items. Used in combination with BeginUpdate;

- `procedure LoadNextBuffer(ABufferSize: Integer = -1);`
  Loads the next list with specific BufferSize. (ABufferSize = -1 loads the default BufferSize).

- `procedure LoadPreviousBuffer(ABufferSize: Integer = -1);`
  Loads the previous list with specific BufferSize. (ABufferSize = -1 loads the default BufferSize).

- `procedure EditMode;`
  Sets the list in edit mode.

- `procedure CancelEditMode;`
  Cancels the edit mode. Sets the list back to normal mode.

- `procedure ShowDetailView;`
  Shows the detail view of the item.

- `procedure HideDetailView;`
  Hides the detail view of the item.

- `procedure LoadBufferBetween(BufferStart, BufferStop: Integer);`
  Loads the list between a start and stop buffer.

- `procedure LookupCustomCategory(ACategoryID: Integer);`
  Automatically positions the category on top in the list with a specific CategoryID. This can be used in combination with custom categories.

- `procedure LookupCategory(ACategory: String);`

Automatically positions the category on top in the list with a specific Category. This can be used in combination with AlphaBetic / AlphaNumeric categories.

- `procedure LookupItem(AItemIndex: Integer; SelectItem: Boolean = False);`
  Automatically positions the item on top in the list with a specific index, and the possibility to select that item.

- `procedure DeSelectAllItems;`
  Deselects all items.

- `procedure SelectAllItems;`
  Selects all items that are loaded in the list.

- `procedure SelectItemsBetween(AStartIndex, AStopIndex: Integer; AddToSelection: Boolean = False);`
  Selects a subset of items between a StartIndex and StopIndex. AddToSelection parameter adds the subset to an internal list, when AddToSelection is false, the list is clear before adding the new subset.

- `procedure SelectItems(Selection: Array of Integer; AddToSelection: Boolean = False);`
  Selects a subset of items for which the item indexes are passed as an array of integers. This procedure works in the same way as the subset between a StartIndex and StopIndex.

Published events

- **OnAfterFilter**: Event called after Filtering is applied.
- **OnAfterSearch**: Event called after Seaching is applied.
- **OnAnimationCustomize**: The transition between the list and the detailview is animated. These 2 list animations can be customized through this event.
- **OnApplyStyleLookup**: When editing the default or custom Style through the StyleBook, this event is called after applying the style. When starting the application, modifications can be made in this event to the style that is either the default style or the edited style from the StyleBook.
- **OnArchiveClick**: Event called when clicking on the ArchiveButton, which is displayed when the TableView is set in EditMode.
- **OnBackButtonClick**: When the TableView is in edit mode, the BackButton is enabled. The OnBackButtonClick event is triggered when clicking on this button.
- **OnBeforeFilter**: Event called before Filtering is applied.
- **OnBeforeSearch**: Event called before searching is applied.
- **OnCategoryAnchorClick**: Event called when clicking on an anchor when using HTML in the Category.
- **OnCategoryClick**: When using categories and the lookupbar is enabled, the OnCategoryClick event is called when clicking on a custom category or an alphanumeric character.
- **OnCategoryCustomize**: When using categories, this event is called after creating a clone of the default category that is available in the default TableView style. Modifications on appearance and functionality can be made through this event.
- **OnHeaderAnchorClick**: Event called when clicking on an anchor when using HTML in the Header.
- **OnFooterAnchorClick**: Event called when clicking on an anchor when using HTML in the Footer.
- **OnItemAfterDetail**: Event called when the animation is completed that animates the detailview after clicking on the item.

- **OnItemAfterDraw**: Event called after the item is drawed. This event can be used for custom drawing on top of the item.
- **OnItemAfterReturnDetail**: Event called when the animation is completed that animates the main list view back to the original state after clicking on the backbutton.
- **OnItemBeforeDetail**: Event called before the animation is started when clicking on an item to animate the DetailView.
- **OnItemBeforeDraw**: Event called before the item is drawn. This event can be used to hide the background and perform custom drawing.
- **OnItemBeforeReturnDetail**: Event called before the animation is started when animating the main list back to the original state after clicking on the backbutton.
- **OnItemCaptionAnchorClick**: Event called when clicking on an anchor when using HTML in the Item Caption.
- **OnItemClick**: Event called when clicking on an item.
- **OnItemCompare**: Event called when sorting the items. Custom sorting can be applied to the items collection.
- **OnItemCustomize**: This event is called after creating a clone of the default item that is available in the default TableView style. Modifications on item appearance, interaction and functionality can be made through this event.
- **OnItemData**: Event called when modyfing one of the Data* properties of an item.
- **OnItemDescriptionAnchorClick**: Event called when clicking on an anchor when using HTML in the Item Description.
- **OnItemDelete**: Event called when deleting an item.
- **OnItemSelected**: Event called when selecting an item.
- **OnManualLoadNextBuffer**: When AutoLoadBuffer is false, this event is called when the list tries to load the next buffer.
- **OnManualLoadPreviousBuffer**: When AutoLoadBuffer is false, this event is called when the list tries to load the previous buffer.
- **OnManualLookup**: When AutoLookup is false, this is event is called when the lookupbar is used to lookup a category.
- **OnMarkClick**: Event called when clicking on the mark button which is displayed when the TableView is set in EditMode.
- **OnMoveClick**: Event called when clicking on the move button which is displayed when the TableView is set in EditMode.
- **OnScroll**: Event called when scrolling in the list.
- **OnScrollFinished**: Event called when the scrolling is finished.

Item storage and buffering

The TableView implements a specific way of loading / displaying items. The design choices are driven by making the TableView performant and resource friendly. Items are stored in a TCollection. When the component is first made visible on a form, from this collection a displaylist of item shapes that are cloned from the default item shape are automatically created. Only a small number of items are created initially. The other remaining items remain in the collection waiting to be shown in the control by means of the generation of display items (shapes). This loading process is necessary to make sure the TableView offers a good performance on various operating systems. In other words, there is no speed impact when adding 10 items or 10000 items to the Items collection. Displayed items are only on demand and incrementally created. This loading / displaying process is controlled by a single property: BufferSize.

The BufferSize property is set, by default, to 50. This means that only for the first 50 items display items are created. While scrolling in the list, the list automatically creates display items for the next 50 items. This buffering method increases performance if balanced correctly. Setting the BufferSize to 0 will create display items at once for all items. Loading 10000 items with BufferSize set to 0 will have an initial large performance hit as the creation of display items is a resource intensive process (FireMonkey framework limitation) Setting BufferSize to 50 is good balance.

If the BufferSize is set, the AutoLoadBuffer property will create the display items automatically when scrolling beyond the end of the list. When AutoLoadBuffer is set to False, the TableView will not create the display items automatically but instead the appropriate events will be triggered. Creating display items can then be done manually.

OnManualLoadNextBuffer: Event triggered if AutoLoadBuffer is false, BufferSize > 0 and the list is scrolled to the end.

OnManualLoadPreviousBuffer: Event triggered if AutoLoadBuffer is false, BufferSize > 0 and the list is scrolled to the beginning.

When implementing these events, loading the buffer can be done with:

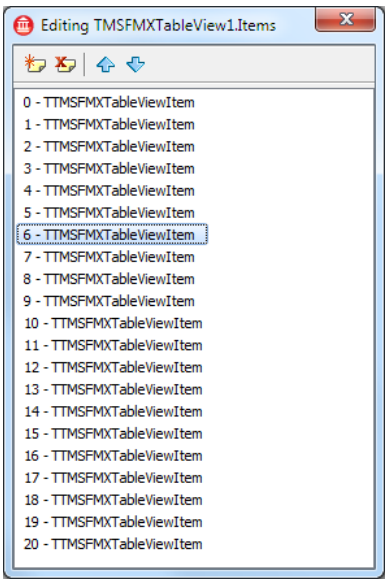
```
TMSFMXTableView1.LoadNextBuffer();
TMSFMXTableView1.LoadPreviousBuffer();
```

These two methods have a parameter with which you can decide how many display items are loaded. If no parameter is passed the loading will use the BufferSize property.

It's a matter of experimenting with the starting BufferSize, the buffersize that is loaded next and the quantity of the items in the collection to have the best performance.

Adding and removing items

After dropping a TableView component (TTMSFMXTableView) on the form you will notice that, by default, the TableView has some items already added. Adding items can be done at designtime and at runtime. Click on the component to view the Items property. This is a default collection editor and can be used to add or remove items.



Items can also be added programatically:

```
var
  it: TTMSFMXTableViewItem;
  i: Integer;
begin
```

```
  TMSFMXTableView1.BeginUpdate;
  for I := 0 to 100 do
  begin
    it := TMSFMXTableView1.Items.Add;
    it.Caption := 'Item ' + inttostr(I);
    it.Description := 'Hello World !';
  end;
  TMSFMXTableView1.EndUpdate;
```

Note that all calls to the TMSFMXTableView must include a BeginUpdate and EndUpdate statement. Whenever an item is added, the display list is rebuilt. With a BeginUpdate and EndUpdate statement the update process is blocked and then executed once.
To remove an item programmatically, just call the item's destructor.

Sorting

After adding items, sorting can be applied to the collection. This will be done alphabetically by default, but can be customized with the OnItemCompare event.

If no sorting is applied, the items are displayed in the order that they are added in the collection.

```
var
  it: TTMSFMXTableViewItem;
  i: Integer;
begin
  TMSFMXTableView1.BeginUpdate;

  it := TMSFMXTableView1.Items.Add;
  it.Caption := 'Pear';
  it.Description := 'This is a Pear';

  it := TMSFMXTableView1.Items.Add;
  it.Caption := 'Banana';
  it.Description := 'This is a Banana';

  it := TMSFMXTableView1.Items.Add;
  it.Caption := 'Apple';
  it.Description := 'This is an Apple';

  it := TMSFMXTableView1.Items.Add;
  it.Caption := 'Lemon';
  it.Description := 'This is a Lemon';

  TMSFMXTableView1.EndUpdate;
end;
```



When calling `TMSFMXTableView1.Items.Sort;` the Items are sorted alphabetically. Two optional parameters can be passed to sort case sensitive (default) or incase sensitive and ascending (default) or descending:

TableView.



Sorting is important for the next paragraph in this manual: Categories.

Categories

In the TableView there are 4 types of categories:

- AlphaBetic
- AlphaNumericFirst (Numbers are added to the lookupbar and are displayed first)
- AlphaNumericLast (Numbers are added to the lookupbar and are displayed last)
- Custom

These types can be set with the CategoryType property. With the AlphaBetic / AlphaNumeric categories, the TableView automatically searches and adds categories with the first letter of the item caption. The range that is used to display categories in AlphaBetic mode is A…Z and AlphaNumeric adds 0…9. In the sample below the items are sorted and the CategoryType property is set to ctAlphaBetic.

```
var
  it: TTMSFMXTableViewItem;
  i: Integer;
begin
  TMSFMXTableView1.BeginUpdate;
  TMSFMXTableView1.CategoryType := ctAlphaBetic;

  it := TMSFMXTableView1.Items.Add;
```

```
  it.Caption := 'Pear';
  it.Description := 'This is a Pear';

  it := TMSFMXTableView1.Items.Add;
  it.Caption := 'Banana';
  it.Description := 'This is a Banana';

  it := TMSFMXTableView1.Items.Add;
  it.Caption := 'Apple';
  it.Description := 'This is an Apple';

  it := TMSFMXTableView1.Items.Add;
  it.Caption := 'Lemon';
  it.Description := 'This is a Lemon';

  it := TMSFMXTableView1.Items.Add;
  it.Caption := 'Apple 2';
  it.Description := 'This is an Apple 2';

  TMSFMXTableView1.Items.Sort;

  TMSFMXTableView1.EndUpdate;
end;
```



When using categories, the LookupBar on the right side of the control is enabled and displayed by default. The LookupBar can be used to navigate directly through the categories. This is explained in the lookup paragraph.

If the CategoryType is set to ctAlphaBetic / ctAlphaNumericFirst or ctAlphaNumericLast the categories are predefined. You can use custom categories by setting the CategoryType to ctCustom and adding your own categories. Custom categories works with linking the CategoryID property of the item to the ID of the category.

```
var
  it: TTMSFMXTableViewItem;
  cat: TTMSFMXTableViewCategory;
  i: Integer;
begin
  TMSFMXTableView1.BeginUpdate;
  TMSFMXTableView1.CategoryType := ctCustom;

  cat := TMSFMXTableView1.Categories.Add;
  cat.Id := 0; // all items with CategoryID 0 belong to this category
  cat.Caption := 'Category 1';
  cat.LookupText := 'Cat 1';

  cat := TMSFMXTableView1.Categories.Add;
  cat.Id := 1; // all items with CategoryID 0 belong to this category
  cat.Caption := 'Category 2';
  cat.LookupText := 'Cat 2';

  it := TMSFMXTableView1.Items.Add;
  it.Caption := 'Pear';
  it.Description := 'This is a Pear';
  it.CategoryID := 0;

  it := TMSFMXTableView1.Items.Add;
  it.Caption := 'Banana';
  it.Description := 'This is a Banana';
  it.CategoryID := 0;

  it := TMSFMXTableView1.Items.Add;
  it.Caption := 'Apple';
  it.Description := 'This is an Apple';
  it.CategoryID := 1;

  it := TMSFMXTableView1.Items.Add;
  it.Caption := 'Lemon';
  it.Description := 'This is a Lemon';
  it.CategoryID := 1;

  it := TMSFMXTableView1.Items.Add;
  it.Caption := 'Apple 2';
  it.Description := 'This is an Apple 2';
  it.CategoryID := 1;

  TMSFMXTableView1.Items.Sort;

  TMSFMXTableView1.EndUpdate;
end;
```



As result you can see the 2 categories in the list and the corresponding lookuptext for each category in the lookupbar. Sorting can also be applied on categories in the same way as sorting is applied to items. The categories are then sorted alphabetically and ascending based on the caption.

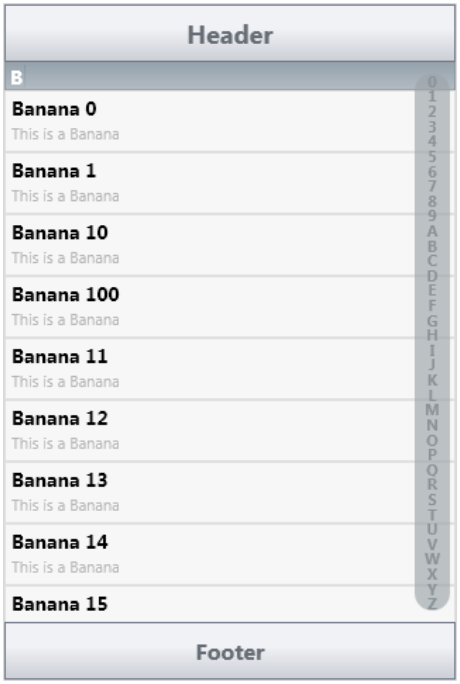
Lookup

After activating categories, the (optional) lookupbar is automatically displayed. Clicking and dragging along the lookupbar will automatically perform a lookup. This can be disabled with the AutoLookup property. If the AutoLookup property is false, the OnManualLookup event is triggered when a category is clicked. From there you can perform a lookup with one of the two methods below depending on the type of categories you have chosen:

```
  TMSFMXTableView1.LookupCategory(ACharacter);
  //Performs a lookup of an AlphaBetic / AlphaNumeric category with a
specific character.

  TMSFMXTableView1.LookupCustomCategory(ACharacterID);
  //Performs a lookup of a Custom category with a specific characterID.
```

When navigating through the categories the lookupbar will, when necessary, load the buffer that is needed to display the selected category. Depending on the BufferSize and the active buffered items the lookup will be either instant or could have a small loading time. Adapt the BufferSize to an optimal value depending on the device on which the application will run.



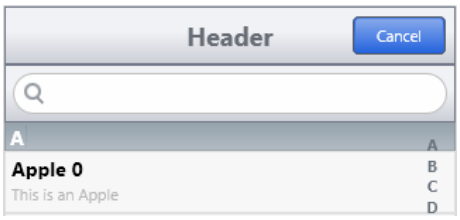
Filtering / Searching

The TableView supports Filtering or Searching. Filtering only shows the items that match the search text while searching highlights the first item found. Similar to the operation of the lookupbar, this way of navigating through items automatically takes care of loading the correct buffer that is needed to display the matched items.

Filtering / Searching can be enabled in two ways: either programmatically with the ShowFilter:Boolean property or by swiping the list down. The last method is optional and can be configured with the ShowFilterOnSwipe property. To programmatically show the filter entry edit control use:

```
TMSFMXTableView1.ShowFilter := True;
```

After setting ShowFilter to true or swiping the list down, the search edit box appears.



The search edit box can be hidden by clicking the cancel button in the TableView header.
By default, when filtering is performed, the TableView only shows the items that match the string in the edit box.
The filter method can be changed by setting the Filtering property and can be set to vfFilterStart, vfFilterRandom, vfSearch or vfNone.

- **vfFilterStart**: Shows the items that match the string from the beginning of the text in the edit box.

- **vfFilterRandom**: Shows the items that contains the filter in the text, not necessarily from the beginning of the text.
- **vfSearch**: Highlights the first item that matches the string in the edit box.
- **vfNone**: No filtering or searching is applied.

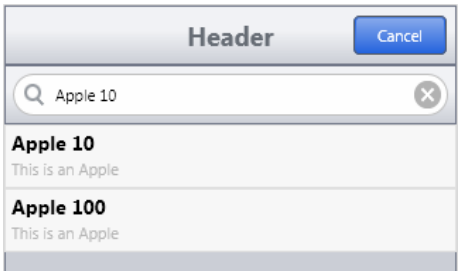
The filtering or searching is automatically performed everytime the text in the search edit box is changed. With the AutoFilter property set to false, this is not longer the case. When AutoFilter is false, the filtering is applied by pressing the Enter key.

Following events are triggered when performing filtering / searching:

OnBeforeFilter, OnBeforeSearch, OnAfterFilter and OnAfterSearch

Via these events, custom filtering or custom searching can be done. To perform a manual filtering you can use the ApplyFilter method:

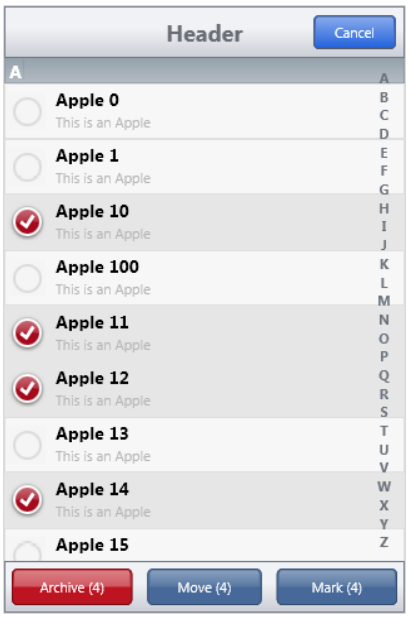
```
TMSFMXTableView1.ApplyFilter('Filter');
```



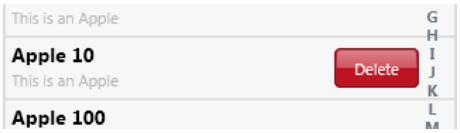
Editing / Deleting

Editing can be enabled with the EditButton: Boolean property. When clicking the edit button, the TableView is set in edit mode. This automatically animates checkboxes from the left and adds buttons in the footer of the TableView. Clicking on the items automatically checks the checkbox and updates the buttons. The text for these editing related buttons can be set with the MarkText, ArchiveText and MoveText properties.





Clicking on the Archive Button will delete the selected items from the list. Each button has a separate event that can be used to perform different operations. Deleting an item can also be done from a delete button associated with the item. The delete button is made visible by swiping from right to left on the item or programmatically:

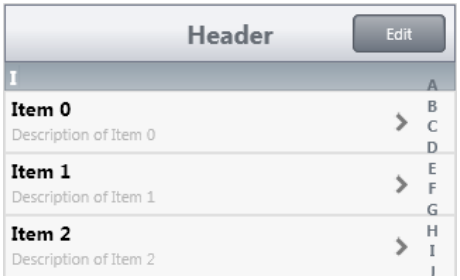


Clicking on the delete button will delete the item. The automatic delete functionality in the deletebutton and the archive button can be disabled with the AutoDeleteItem property.

Note: Editing and searching are mutually exclusive functions. When searching is enabled the edit functionality is disabled and vice versa.

DetailView

The TableView has support for showing a Detailview. The detail view can be any type of component that descends from TControl. Each item in the TableView can have its own DetailView or all items can share the same DetailView (DetailView property on item level versus DefaultDetailView on TableView level). If the DetailView is assigned, the item is marked with a default detail arrow image. If the DetailView property is nil, the item automatically looks for the DefaultDetailView control that can be assigned on the TableView level.

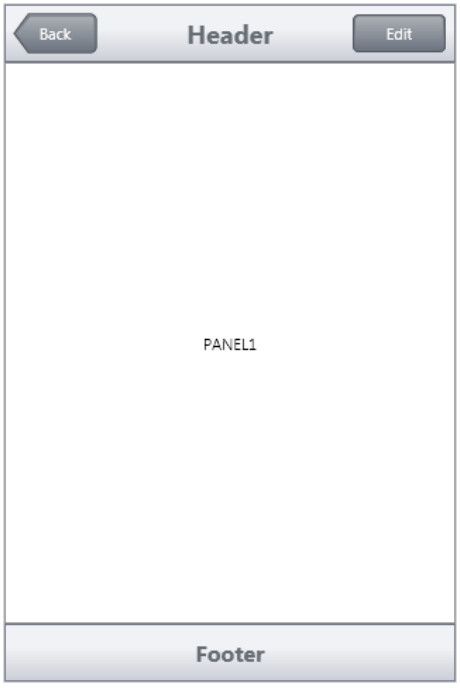


Clicking on the item that has a DetailView assigned automatically slides the main items list out of view and slides in the DetailView. An extra back button, controlled by the TableView BackButton property can be optionally shown and when clicked, it will slide out the DetailView and slide in the regular main items list.

In this sample, we have placed three panels on the form and assigned each panel to a different item:

```
TMSFMXTableView1.Items[0].DetailView := Panel1;
TMSFMXTableView1.Items[1].DetailView := Panel2;
TMSFMXTableView1.Items[2].DetailView := Panel3;
```

Clicking on the first item shows Panel1, the second item Panel2 and the third item Panel3.



The automatic toggle between the main list and the detail list can be disabled by setting AutoToggleDetail: Boolean to false. From the OnItemClick or OnItemSelected event handler you can use the ToggleDetailView() method to show the assigned DetailView of the item. With the OnBackButtonClick event you can programmatically return back to the main list.

```
TForm1.TMSFMXTableView1BackButtonClick(Sender: TObject;
  AItem: TTMSFMXTableViewItem);
```

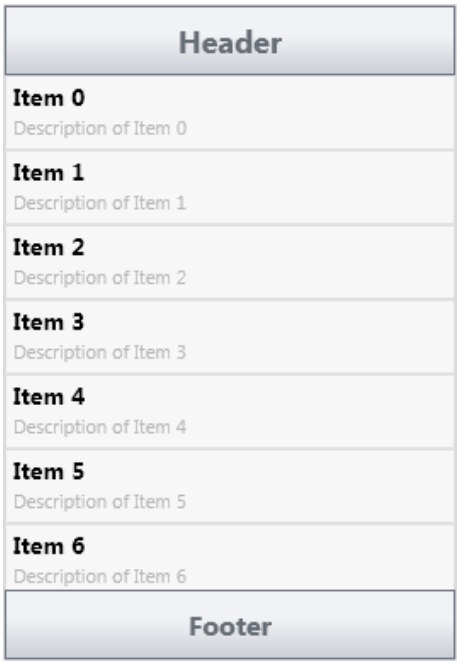
```
begin
  AItem.ToggleDetailView;
end;

TForm1.TMSFMXTableView1ItemClick(Sender: TObject;
  AItem: TTMSFMXTableViewItem);
begin
  AItem.ToggleDetailView;
end;
```

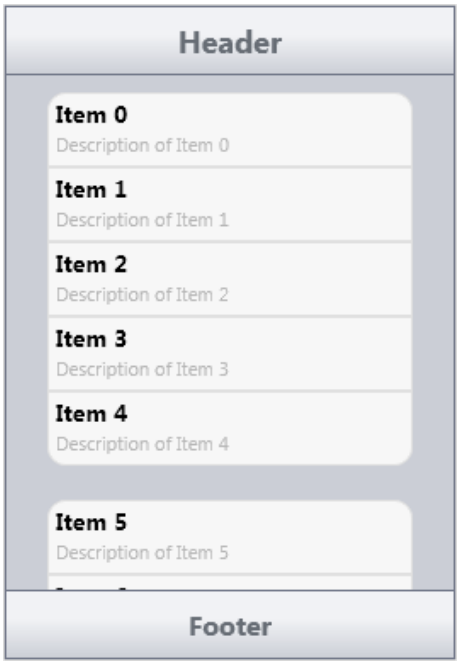
Layout

The TableView has 2 ways of displaying the list. The list can either be displayed in a normal view (default) or in a grouped view. This can be selected with the TTMSFMXTableView.LayoutMode property.

Normal view:



Grouped view:



Items are assigned to a group (Category) via the Item's CategoryID property. When using categories, multiple items can be assigned to a group (Category).

This sample code snippet initializes two groups with each two items in the TableView:

```
var
  itm: TTMSFMXTableViewItem;
begin
  TMSFMXTableView1.BeginUpdate;
  TMSFMXTableView1.Items.Clear;

  TMSFMXTableView1.Categories.Clear;
  // group 0
  TMSFMXTableView1.Categories.Add.Caption := 'Internet settings';
  // group 1
  TMSFMXTableView1.Categories.Add.Caption := 'Email settings';

  TMSFMXTableView1.CategoryType := ctCustom;

  itm := TMSFMXTableView1.Items.Add;
  itm.CategoryID := 0;
  itm.GroupIndex := 0;
  itm.Caption := 'Proxy';
  itm.Description := 'Proxy server name';

  itm := TMSFMXTableView1.Items.Add;
  itm.CategoryID := 0;
  itm.GroupIndex := 1;
  itm.Caption := 'IP address';
  itm.Description := '192.168.0.1';
```

```
  itm := TMSFMXTableView1.Items.Add;
  itm.CategoryID := 1;
  itm.Caption := 'POP server';
  itm.Description := 'pop.myserver.com';

  itm := TMSFMXTableView1.Items.Add;
  itm.CategoryID := 1;
  itm.Caption := 'SMTP server';
  itm.Description := 'smtp.myserver.com';


  TMSFMXTableView1.LayoutMode := lmGroup;
  TMSFMXTableView1.EndUpdate;

end;
```



As you can see with in the above result, the items from the first category are added separately. This can be controlled with the GroupIndex property. Items can be added to different categories with the CategoryID and within the category, multiple groups can be created with the GroupIndex property.

User interface interaction with the TableView

The list supports scrolling, navigation and selection. Interaction with the list can be done with the keyboard and the mouse.

With the keyboard you can navigate through the items, each time you press the up, down, left or right keys the list selects the previous or next item. The home key selects the first item while the end key selects the last item. When pressing the pagedown or pageup key the list jumps in steps of the BufferSize, if the BufferSize = 0 the step is set to 10.

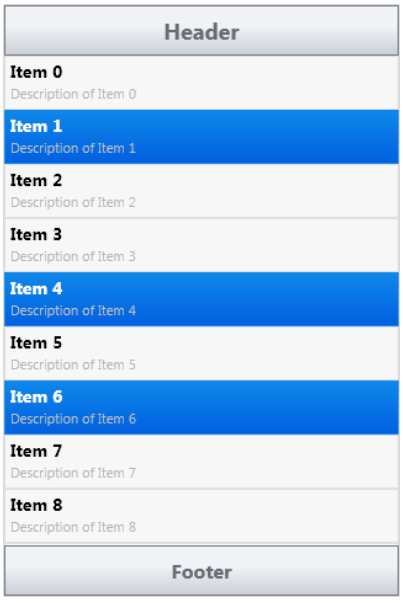
When navigating through the list with the keyboard, the list automatically loads the next buffer.

To navigate in the list with the mouse, you can click and hold your left mouse button on the list then drag up or down depending in the direction you want the list to scroll. When making a flick gesture the list will scroll a certain amount of items with an inertia animation. The faster you flick with the mouse the faster and the further the list will scroll.

MultiSelect

The TableView supports multiple item selection. Multiple item selection can be enabled by setting the MultiSelect property to true. Clicking on an item selects that item and when you click on another item, the previous item is deselected. To select multiple items, you need to hold the CTRL or SHIFT key on the keyboard. With the CTRL key you are able to select items of choice and with the SHIFT key the list selects all items between the first and the last item you have clicked.

MultiSelect can be done by keyboard alone, pressing the direction keys and holding the CTRL or SHIFT key at the same time. Below is a sample of multiple items selected in the list by holding the CTRL key and clicking several items.



Programmatically, in multiselect mode, the selected of an item can be get or set with the item's Selected: Boolean property. In addition, two helper methods exist: SelectAllItems, DeSelectAllItems to programmatically select or unselect all items at once.

This sample code snippet programmatically selects items 1,3,5:

```
TMSFMXTableView1.MultiSelect := true;
```

```
TMSFMXTableView1.DeSelectAllItems;
TMSFMXTableView1.Items[1].Selected := true;
TMSFMXTableView1.Items[3].Selected := true;
TMSFMXTableView1.Items[5].Selected := true;
```

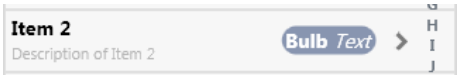
Additional Item Elements

Other than the delete button, the detail arrow image and the checked / unchecked image, the item has three additional elements that are optionally (See notes in Performance chapter) available:

- LeftRectangle
- RightRectangle
- BulbRectangle

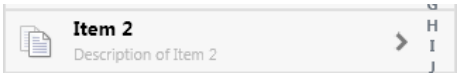
The BulbRectangle contains a TTMSFMXHTMLText shape that contains optional HTML formatted Text and can be set with the BulbText property.

```
TMSFMXTableView1.Items[0].BulbText :='<b>Bulb</b> <i>Text</i>';
```



The Left Rectangle contains a TTMSFMXBitmap component that can be linked to a BitmapContainer. By default the Left Rectangle is disabled. You can enable it by checking the correct option in the ItemOptions property.

Note that an extra area is offered where a bitmap can be shown. You can either show a BitmapContainer item and link it via the BitmapName, or directly load the bitmap in the item.




```
var
  bmp: TTMSFMXBitmapItem;
begin
  bmp := TMSFMXTableView1.Items.Add;
  bmp.Name := 'item_bitmap';
  bmp.Bitmap.LoadFromFile('mybitmap.png');

  TMSFMXTableView1.BitmapContainer := TMSFMXBitmapContainer1;
  TMSFMXTableView1.Items[0].BitmapName := 'item_bitmap';
  //or
  TMSFMXTableView1.Items[0].Bitmap.LoadFromFile('mybitmap.png');
end;
```

The right rectangle is an empty rectangle by default and can contain any visual element that is available in FireMonkey. The TableView supports a small set of controls, that are ready to use and do not need further binding with properties and events. This is explained in the next chapter.

Performance

A default TableView has some performance related options already preset to optimal values for the most typical use of the TableView. The BufferSize is set to 50, which will only load and display the first 50 items in the UI. Second performance related setting is the ItemOptions property. This contains a set of options that can be used to decrease the time the TableView needs to build up the display items list.

As explained in the Architecture chapter, the default item contains various elements that are cloned when adding items. These setting under ItemOptions affect what parts of an item are cloned or not. When an element is not checked in the ItemOptions, the element is not cloned when the display item list is built and that increases the performance.

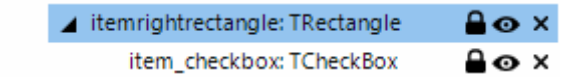
By default the ItemOptions have unchecked LeftRectangle, RightRectangle and BulbRectangle. This means that these elements will not be cloned and will not be available in the item. You can further improve performance by unchecking elements that are not necessary for your application.

Binding Controls

When the right rectangle is activated in the ItemOptions, an extra space will be available that can be filled with the control added in the StyleBook. You can add any type of control, but the list already implements a small set that does not need extra binding.

- CheckBox
- RadioButton
- TrackBar
- ArcDial

Open the StyleBook editor by clicking on the Edit Custom Style option in the popupmenu after right-clicking on the component. Select the default item -> right rectangle element and place a new TCheckBox component inside the rectangle.



Select the TCheckBox component and set the StyleName to ‘item_checkbox’. When adding a RadioButton, TrackBar or ArcDial, the component base classname – ‘T’ must be prepended with ‘item_’. The StyleName must be lowercase.

- CheckBox: ‘item_checkbox’
- RadioButton: ‘item_radiobutton’
- TrackBar: ‘item_trackbar’
- ArcDial: ‘item_arcdial’

Applying the style in the IDE style editor results in the Checkbox being visible at designtime in each item in the list. In the TableView, the CheckBox OnChange event has been assigned and the IsChecked property is linked to the Item from the Items collection.

To persist the value of the checkbox, the Item has a few Data* properties to store the value:

- DataBoolean
- DataGroupIndex
- DataString
- DataValue
- DataObject

For the CheckBox and RadioButton the DataBoolean property is used, for the TrackBar and ArcDial the DataValue property is used. When changing the DataBoolean property to true, the checkbox will be checked. When checking the checkbox, the DataBoolean property will be set.

This binding happens in the same way for the other controls. For the RadioButton the Item has an extra DataGroupIndex property that can be used to create a RadioGroup, i.e. all items with a RadioButton with the same DataGroupIndex value form a group.

Manual binding is also available through events for non-supported controls. Below is a sample binding a TEdit to the DataString property of the item.

First, add a TEdit control to the right rectangle element in the StyleBook. Set the TEdit control stylename to 'item_edit'. This will be important to access the edit with the correct name. Second, there are 3 important events that are needed for binding with TEdit.

- OnItemCustomize
- OnItemData
- OnChange

The OnItemCustomize event is used to connect the edit with the DataString property. The OnItemData is called when the Data properties are modified. The OnChange event is implemented and sets the DataString property with the value of the edit. The combination results in the code below:

```
TForm1.ItemEditChanged(Sender: TObject);
var
  i: Integer;
  it: TTMSFMXTableViewItem;
begin
  i := (Sender as TEdit).Tag;
  if (i >= 0) and (i <= TMSFMXTableView1.Items.Count - 1) then
  begin
    it := TMSFMXTableView1.Items[i];
    it.DataString := (Sender as TEdit).Text;
  end;
end;

TForm1.TMSFMXTableView1ItemCustomize(Sender: TObject;
  AItem: TTMSFMXTableViewItem; AItemShape: TTMSFMXTableViewItemShape;
  AItemControlShape: TControl);
var
  shpr: TRectangle;
  edt: TEdit;
begin
  shpr := AItem.ShapeRightRectangle;
  if Assigned(shpr) then
  begin
    edt := (shpr.FindStyleResource('item_edit') as TEdit);
    if Assigned(edt) then
    begin
      edt.OnChangeTracking := ItemEditChanged;
      edt.Tag := AItem.Index;
    end;
  end;
end;
```

```
TForm1.TMSFMXTableView1ItemData(Sender: TObject;
  AItem: TTMSFMXTableViewItem; AItemShape: TTMSFMXTableViewItemShape;
  AItemControlShape: TControl);
var
  shpr: TRectangle;
  edt: TEdit;
begin
  shpr := AItem.ShapeRightRectangle;
  if Assigned(shpr) then
  begin
    edt := (shpr.FindStyleResource('item_edit') as TEdit);
    if Assigned(edt) then
      edt.Text := AItem.DataString;
  end;
end;
```

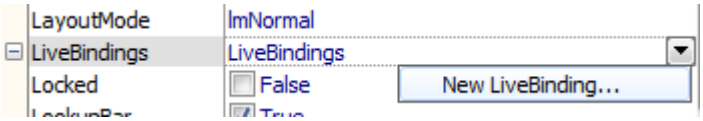
LiveBindings

LiveBindings is new way of data-binding controls, for more information and samples on LiveBindings, please read the documentation on http://docwiki.embarcadero.com/RADStudio/en/LiveBindings_in_RAD_Studio first.

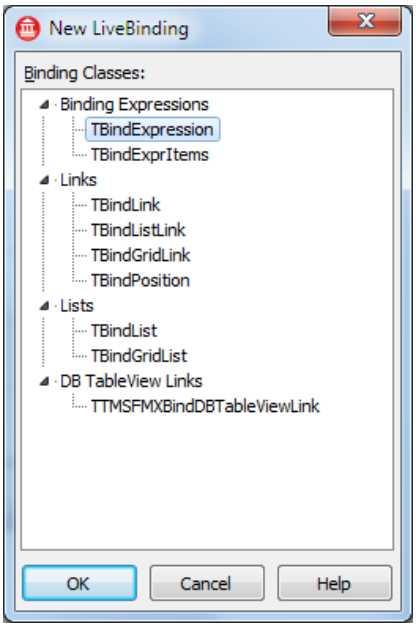
The TableView has a basic implementation that populates the list and takes care of binding the SelectedItemIndex. When dropping a TableView on the form you will notice a LiveBindings property.



To create a new LiveBinding, you can either click on the arrow and select “New LiveBinding…” or click directly on “New LiveBinding…” at the bottom of the object inspector.

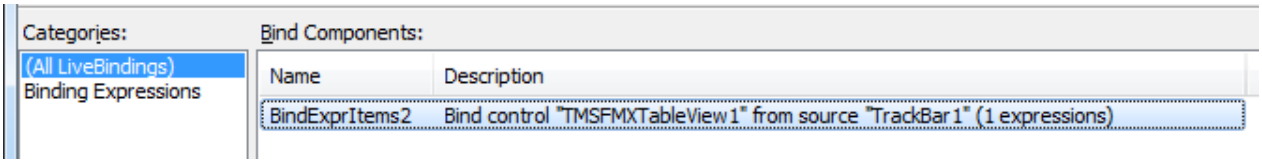


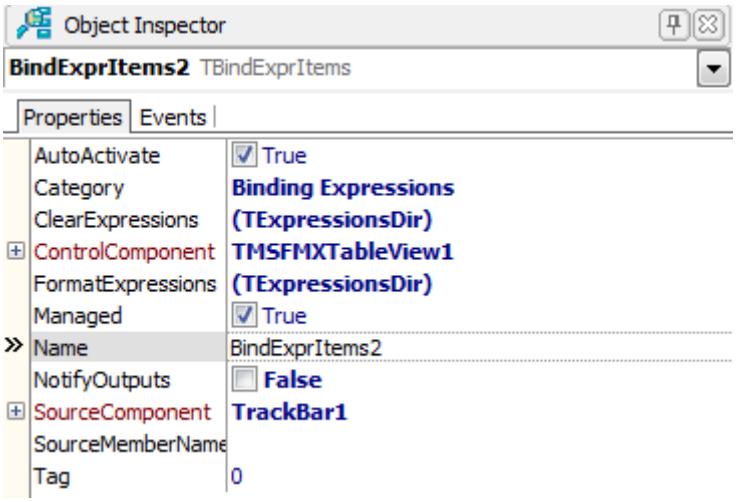
This action will automatically drop a BindingsList component on the form and will show the BindingsList editor window.



When reading the documentation in the “LiveBindings in RAD Studio” you will notice that LiveBindings is not limited to DataBase support. There is also support for binding various properties of the TableView to other controls. Below is a sample that binds the TrackBar position to the TableView rotationAngle.

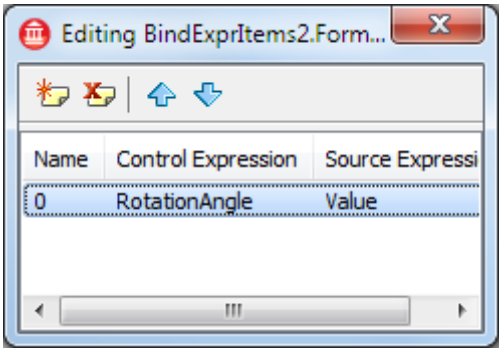
Drop a new TableView and a TrackBar component on the form. Select the TrackBar component and add a new TBindExprItems expression. In the BindingsList component you see the TBindExprItems component listed.





Point the SourceComponent to the TrackBar and the ControlComponent to the TMSFMXTableView1. Start the FormatExpressions editor by double-clicking on the “(TExpressionsDir)” label. Click on the add button to add a new format expression and fill in the Binding properties. The ControlComponent

is set to TMSFMXTableView1 and we want the RotationAngle to be modified. Fill in RotationAngle in the Control Expression field. For the Source Expression which is linked with the SourceComponent we fill in “Value”.



Now there is one step left to implement before the application is ready. When dragging the slider of the TrackBar, the RotationAngle of the TMSFMXTableView component will be unaffected. This is because the TMSFMXTableView did not receive a notice from the BindingsList component, therefore we need to notify the bind component that the value of the TrackBar has changed.

This is done by implementing the OnChange event and notifying the BindingsList component:

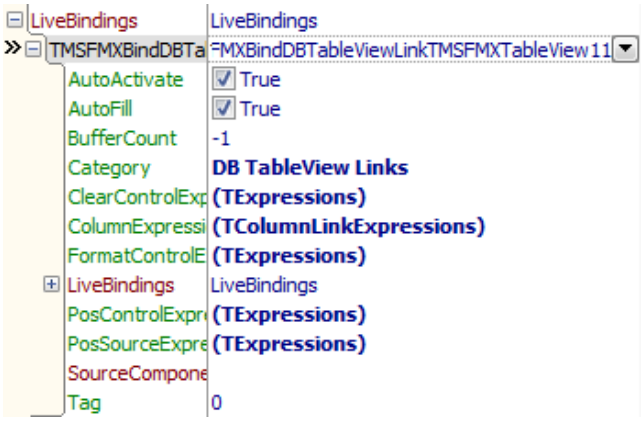
```
var
  FNotifying: Integer;

procedure TForm557.TrackBar1Change(Sender: TObject);
begin
  // Some controls send notifications when setting properties,
  // like TTrackBar
  if FNotifying = 0 then
  begin
    Inc(FNotifying);
    // Send notification to cause expression re-evaluation of dependent
expressions
    try
      BindingsList1.Notify(Sender, '');
    finally
      Dec(FNotifying);
    end;
  end;
end;
```

Now when dragging the slider of the TrackBar, the BindingsList is notified and the TMSFMXTableView rotates according to the Value of the TrackBar. Multiple bindings can be made and are triggered simultaneously due to the Notify procedure of the BindingsList.

In the BindingsList editor window you will notice that a new category is added for the TableView: “DB TableView Links”. The link “TTMSFMXBindDBTableViewLink” can be used to connect to a DataSource through a BindScope. This link is designed to work specifically with DataBase connections.

The TableView DB link automatically generates Expressions to link the SelectedItemIndex to the correct record in the Database. Internally the DataSet is automatically loaded and for each record in the DataSet an item is added.



In the object inspector, after expanding the LiveBindings property, you will notice a ColumnExpressions property. This is because the implementation inherits from the standard TBindGridLink and is necessary to link Database fields to multiple elements in the TableView.

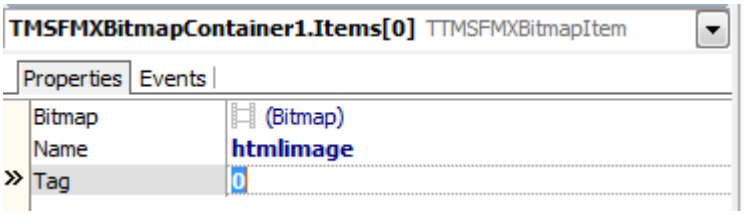
More on the TTMSFMXBindDBTableViewLink is explained in the Samples chapter at the TMS TableView LiveBindings Demo 1 & 2.

HTML support

The TableView supports HTML in various elements such as the headertext, footertext, bulbtext, the caption and the description. The HTML engine in FireMonkey is a port of the VCL HTML engine, thus supporting most functionality that was available in the VCL HTML Engine.

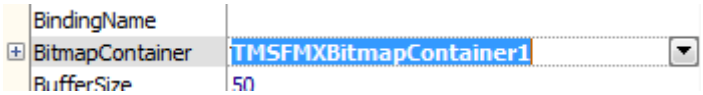
As ImageLists do no exist in the FireMonkey framework, the HTML engine supports displaying images from a BitmapContainer and this is demonstrated in the sample below.

Drop a TableView and a BitmapContainer on the form. In the BitmapContainer we have loaded an image and named it htmlimage:



The name is necessary because the HTML Engine will search for an image with a name that is set in the <img> tag.

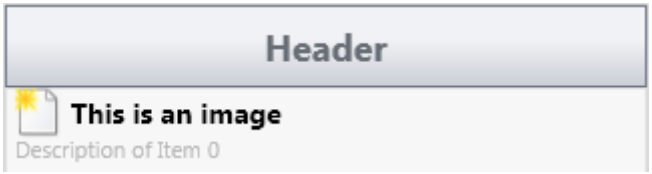
Now assign the BitmapContainer to the TableView component:



In this sample we will display an image in HTML in the caption of an item.
This can be done by using the <img> tag.

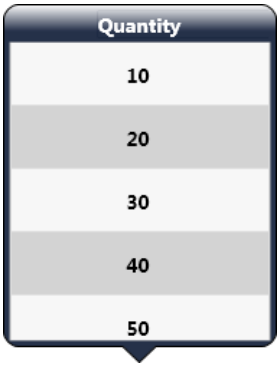
```
  TMSFMXTableView1.Items[0].Caption := '<img src="htmlimage"> This is an
image';
```

The item will display the caption with an image that comes from the BitmapContainer. In this way, one item can be used multiple times.



For more information about the HTML engine and supported tags please read the chapter HTML rendering engine.

TTMSFMXPopup



The TTMSFMXPopup is a component that has the capability to display a FireMonkey control inside a fully customizable transparent popup window. It is specifically designed match the layout of a native iOS popup window. This component can be easily configured to display itself positioned at a specific control on the form or a given absolute position. The popup control can have optionally a footer and a header and in the footer and header buttons can be added as well as HTML formatted text.

Header and footer are configurable via following properties:

- **FooterButtons**: collection of TTMSFMXPopupButton instances displayed from left in the footer
- **FooterHeight**: sets the height in pixels of the footer
- **FooterText**: string holding HTML formatted text for the footer
- **HeaderButtons**: collection of TTMSFMXPopupButton instances displayed from left in the header
- **HeaderHeight**: sets the height in pixels of the header
- **HeaderText**: string holding HTML formatted text for the header

Following properties exist for the TTMSFMXPopupButton:

- **BitmapName**: name of image in the connected BitmapContainer to use for the button
- **Kind**: sets the type of the button, following button types are predefined: bkArchive, bkBack, bkCancel, bkDelete, bkDone, bkEdit, bkMark, bkMove, bkNormal
- **Layout**: property to set layout of the button to normal or pointer
- **ShowText**: when true, text is shown next to the image in the button
- **Text**: button caption text
- **Width**: width of the button

Properties on TTMSFMXPopup that control appearance and position of the popup:

- **ArrowHeight**: The height of the arrow.
- **ArrowPosition**: The position of the arrow.
- **ArrowWidth**: The width of the arrow.
- **DetailControl**: The control that will be displayed and automatically positioned inside the popup.
- **HeaderText**: The text of the header that supports HTML.
- **Placement**: Property to determine the position of the popup relative or absolute to a given placement control.

- **PlacementTarget**: The control used as a reference to position the popup after it is displayed.
- **PlacementRectangle**: A custom rectangle used to display the popup, in combination with the PlacementTarget and Placement properties.

After setting properties where the popup must be shown you can use the following methods to popup or close the dialog:

```
TMSFMXPopup1.Popup;
TMSFMXPopup1.Close;
```

Example:

This code snippet assigns a TTMSFMXTableView control to be displayed on the popup and configures the TTMSFMXPopup to open at the bottom of a button on the form with a button in the header to close the popup again:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // hide the popup footer
  TMSFMXPopup1.ShowFooter := false;
  // tableview should be displayed in the popup only
  TMSFMXTableView1.Visible := false;
  // assign the tableview as detail control for the popup
  TMSFMXPopup1.DetailControl := TMSFMXTableView1;
  // set the control as reference for position of the popup
  TMSFMXPopup1.PlacementTarget := Button1;
  // show the popup at the bottom of the button centered
  TMSFMXPopup1.Placement := TPlacement.plBottomCenter;
  // add a header button that closes the popup
  TMSFMXPopup1.HeaderButtons.Clear;
  with TMSFMXPopup1.HeaderButtons.Add do
  begin
    Kind := TTMSFMXBarButtonKind.bkBack;
    Width := 50;
  end;
end;

procedure TForm1.TMSFMXPopup1HeaderButtonClick(Sender: TObject;
  AButton: TTMSFMXPopupButton; AButtonShape: TTMSFMXBarButton);
begin
  if AButton.Index = 0 then
    TMSFMXPopup1.Close;
end;
```

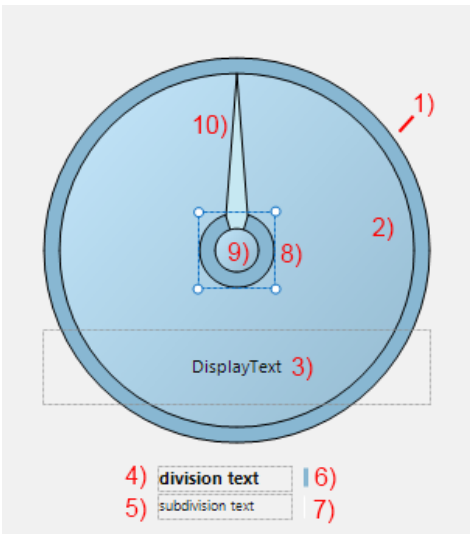
TTMSFMXCircularGauge

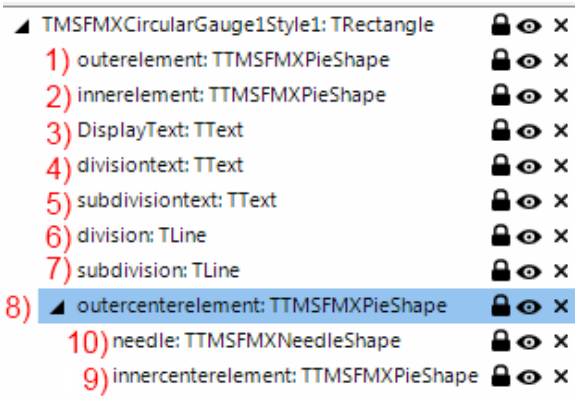
TTMSFMXCircularGauge is a highly customizable circular gauge with optionally multiple needles, setpoints and sections. Divisions, subdivisions can be configured as well as aperture of the values or aperture of the entire control, enabling to use it as a full circle gauge, half circle gauge, ¾ circle gauge etc…

Customizing the appearance of a FireMonkey component is done via editing its style.

Design of the TTMSFMXCircularGauge

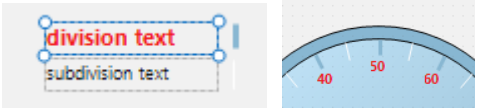
Following style elements make up the appearance of the TMSFMXCircularGauge:





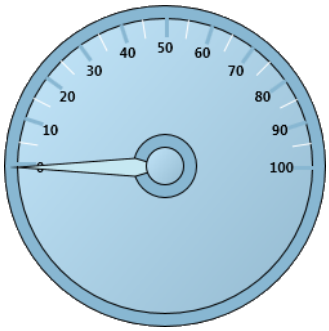
1) The outer background element of the gauge.
2) The inner element of the gauge. This element holds the sections, needles, setpoints, divisions and subdivisions.
3) The displaytext of the gauge, this can be set on component level.
4) 5) The Division and SubDivision text appearance.
6) 7) The Division and SubDivision line appearance.
8) The outer center element.
9) The needle.
10) The inner center element which holds the needle.

When starting the IDE style editor for the gauge, there are two TText elements and two TLine elements (item 4 till 7). These elements are used to define the appearance of the Divisions and the SubDivisions in the gauge. The style of these elements will be reflected in the control when applying the style. A single style element for the Division and SubDivision will control the appearance of each divider line/text and subdivider line text in the gauge.

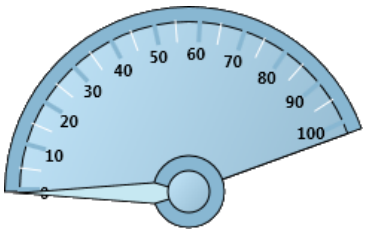


By default the Divisions and SubDivisions are visible. The properties ShowDivisions, ShowSubDivisions control this visibility. In the Object Inspector, the component has properties that can be used to specify the number of Divisions / SubDivisions, change the minimum, maximum and the value. The values are distributed in the circular gauge between a start and stop angle that can be set with DivisionsStartAngle, DivisionsStopAngle. For the gauge itself, the LayoutStartAngle, LayoutStopAngle define the start angle and end angle of the shape and these are default set to 0° and 360° to have a full circular gauge. Set these values to 0°, 180° for example to have a half circular gauge.

Division angles:

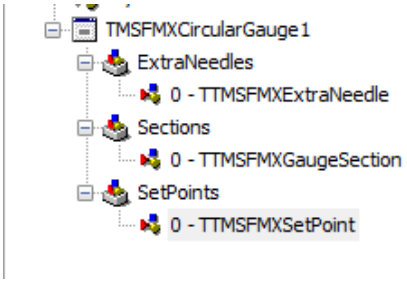


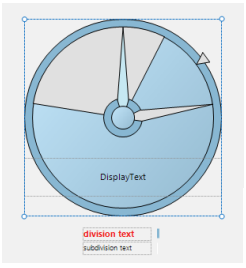
Layout angles:

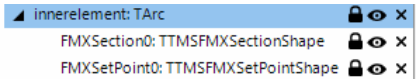


Changing the Value property of the gauge will update the needle position accordingly.

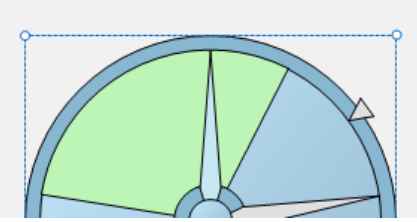
The gauge has three different collections of shapes that will be dynamically added to the StyleBook when used and can each be modified afterwards. Adding a setpoint, section or extra needle will automatically add an element in the StyleBook.







Remember that the position of the extra needle and the setpoint, or the start value and end value of the section, are properties that are accessible in the IDE Object Inspector. The visual configuration of these setpoints, sections or extra needles can be changed with the IDE style editor and will determine the way these elements look.



You can style each element separately, add or remove elements and create a custom version of the default gauge. Below is a sample of the gauge after adding new section elements, and modifying the background, inner- and outerelement appearances.



Other than via the designer, a section for example can also be added in code:

Example:

This code snippet adds a yellow section for the range of values from 20 to 40:

```
with TMSFMXCircularGauge1.Sections.Add do
begin
  StartValue := 20;
  EndValue := 40;
  Shape.Fill.Color := claYellow;
end;
```

Programmatically changing style elements

In code, the different style elements can also be changed. The component exposes these style elements via functions. For example, TTMSFMXCircularGauge provides the function GetDivision, GetDivisionText etc…

In code, the color of the divider lines could be changed with:

```
TMSFMXCircularGauge1.GetDivision.Fill.Color := claYellow;
TMSFMXCircularGauge1.GetDivision.Stroke.Color := claYellow;
TMSFMXCircularGauge1.Update; //needed to update the style!
```

Events

- **OnValueClick**: Event triggered when the inner element of the gauge is clicked and passes the value at the mouse coordinate.
- **OnSectionClick, OnSetPointClick, OnExtraNeedleClick**: Events triggered when clicking on section, setpoint or needle elements. When implementing the OnSectionClick event handler, clicking a section will not trigger the OnValueClick since the section lays on top of the values.

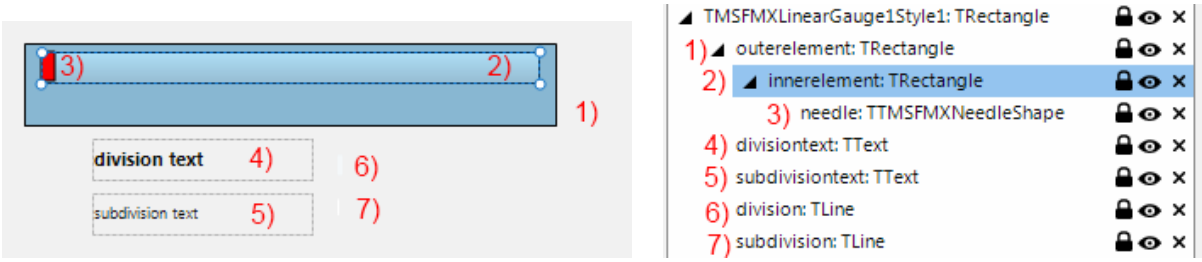
Properties

- **DivisionFormat**: The format of the main divisions in the gauge, formatting is done internally with the FormatFloat() function. See FormatFloat in the Delphi help for information on the available formatting specifiers.
- **Divisions**: The amount of divisions between MinimumValue and MaximumValue.
- **DivisionsStartAngle / DivisionsStopAngle**: The start and end angles between which the dividers are distributed in the gauge inner circle.
- **ExtraNeedles**: Extra needles can be added in this collection, the needles can be styled in the style editor and the position can be set with the Value property of each needle.
- **LayoutStartAngle / LayoutStopAngle**: The start and stop angle used to display the gauge. By default this is 0°and 360° to display the gauge as a full circle.
- **MinimumValue / MaximumValue**: Defines the minimum and maximum for the values displayed in the gauge. The number of values displayed is determined by the minimum, maximum and the number of divisions and subdivisions.
- **Sections**: A collection of elements used inside the gauge to mark special areas or ranges of values. Sections start from the center but the size can be set with the inner- and outermargin properties.
- **SetPoints**: SetPoints are used to mark a special value, and are placed at the outer border of the gauge. SetPoints can have different shapes and can be styled, in the same way as the extra needles and the sections.
- **ShowDivisions / ShowSubDivisions / ShowDivisionText / ShowSubDivisionText**: Properties used to hide / show values and divider lines on the gauge.
- **SubDivisionFormat**: Sets the format of the subdivisions. Subdivisions are formatted in the same way as the Divisions.
- **SubDivisions**: Defines the amount of divisions between 2 main divisions.

TTMSFMXLinearGauge

TTMSFMXLinearGauge is very similar to the TTMSFMXCircularGauge. It also offers a needle, divisions and subdivisions, optional sections, setpoints or extra needles.

Design of the TTMSFMXLinearGauge

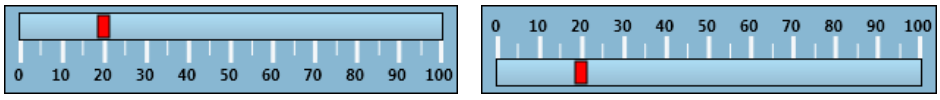


1) The outer background element of the gauge.
2) The inner element of the gauge. This element holds the sections, needles, setpoints, divisions and subdivisions.
3) The needle of the gauge.
4) 5) The Division and SubDivision text appearance.
6) 7) The Division and SubDivision line appearance.

The linear gauge is internally based on the circular gauge, but implements a different style to display it in a rectangular way. The SetPoint, Section and ExtraNeedle collections are also available in the TTMSFMXLinearGauge. The SetPoints are clickable in the circular gauge and thus also in the linear gauge. When clicking on a setpoint, the OnSetPointClick event is triggered.

When adding setpoints, sections or extra needles, the shapes are visible in the IDE style editor in the same way as with the circular gauge.

With the ValueDirection property you can control whether the divider lines and values are displayed on top of or below the gauge.

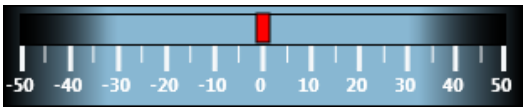


Modifying the style of the TTMSFMXLinearGauge to display vertical

Set TTMSFMXLinearGauge.RotationAngle to 90 and via the IDE style editor, change the divisiontext style resource's OrientationAngle to -90. That is all that is needed to create a vertical variant of the TTMSFMXLinearGauge. By default the values are displayed now on the left of the gauge, but this can be easily changed by setting TTMSFMXLinearGauge.ValuePosition = vpUp.

TTMSFMXJogMeter

The jogmeter is a similar component as the TTMSFMXLinearGauge but has no minimum and maximum values. The range is defined via a combination of Divisions, Aperture and Step. The needle always remains in the center of the control and the displayed range changes if the value is set.

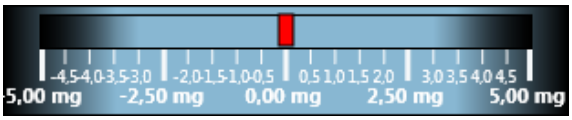


This component descends from the TTMSFMXLinearGauge component and provides a jogmeter specific implementation.

This code sample shows how the TTMSFMXJogMeter can be setup for a specific range of values:

```
TMSFMXJogMeter1.Step := 0.1;
TMSFMXJogMeter1.DivisionFormat := '0.00 mg';
TMSFMXJogMeter1.Divisions := 25;
TMSFMXJogMeter1.ShowSubDivisionText := True;
TMSFMXJogMeter1.SubDivisions := 5;
```

The innerelement and the subdivision line was also made slightly smaller to ensure all values fit. This can be done in the IDE style editor. After applying this code and changing the innerelement the gauge should look like the sample below:

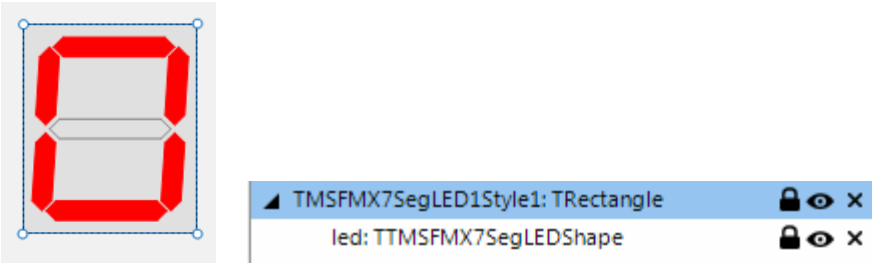


Note that the same techniques used to create a vertical variant of the TTMSFMXLinearGauge can be applied to create a vertically oriented TTMSFMXJogMeter.

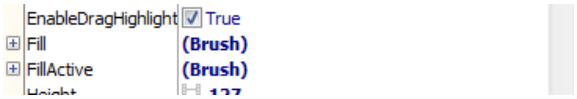
TTMSFMX7SegLED

TTMSFMX7SegLED is a control to display a value via 7 segment LEDs.

Style:



The 7-segment LED uses one LED shape that can be used to change the appearance, the fill of the active and non-active LED segments. The Fill and FillActive brushes can be accessed via the IDE style editor after clicking on the LED shape.



The 7-segment LED control has a Decimals and Digits property, which are used to increase or decrease the amount of 7-segment LEDs that are visible. Digits sets the number of 7-segment LEDs before the decimal point and Decimals sets the number of 7-segment LEDs after the decimal point. The Value property is used to set the number that must be displayed. Note that the control will not automatically adapt its width when the number of digits or decimals is changed. It will try to fit all 7-segment LEDs within the available space. Modify the width of the control when needed.

```
var
  val: double;
begin
  TMSFMX7SegLED1.Digits := 4;
  TMSFMX7SegLED1.Decimals := 2;
  val := 1234.56;
  TMSFMX7SegLED1.Value := val;
end;
```

Programmatically, the style of the LED can be changed via accessing the shape:

```
TMSFMX7SegLED1.Shape.FillActive.Color := claGreen;
TMSFMX7SegLED1.Update;
```

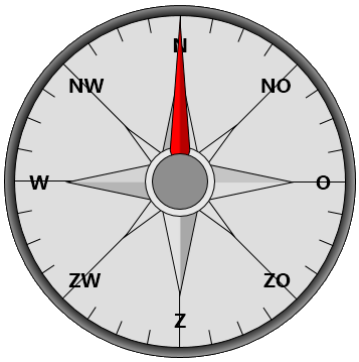
TTMSFMXCompass

The compass control descends from the gauge base class. It can be considered as a gauge with the needle pointing to the north and 8 extra needles for each wind direction. These needles are accessible in the style editor as:

needle: needle pointing to the north
North_Needle: background needle in north direction
South_Needle: background needle in south direction
West_Needle: background needle in west direction
East_Needle: background needle in east direction
NorthWest_Needle: background needle in northwest direction
SouthWest_Needle: background needle in southwest direction
NorthEast_Needle: background needle in northeast direction
SouthEast_Needle: background needle in southeast direction

The DivisionText style is used here to define the style of the wind direction texts in the compass.

The orientation of the compass can then be easily modified with the TTMSFMXCompass.RotationAngle property.



TTMSFMXClock

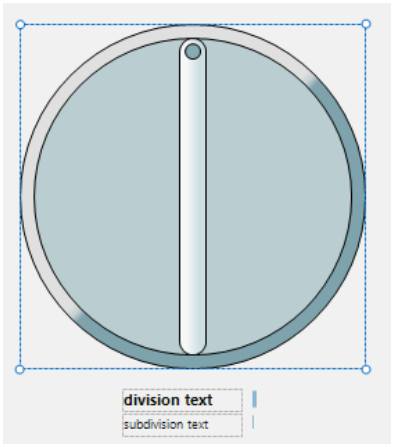
The clock can display the current machine time or can display the time set by the ClockTime property. When the Active property is true, it displays the machine time. Otherwise, the time shown on the clock can be set via code:

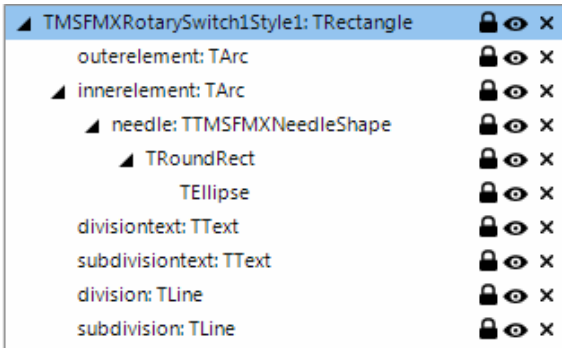
```
TMSFMXClock1.ClockTime := EncodeTime(3,15,20,0);
```

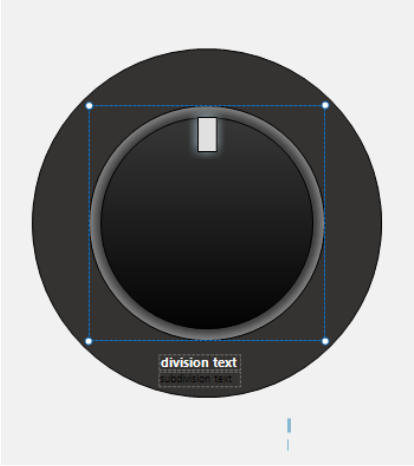


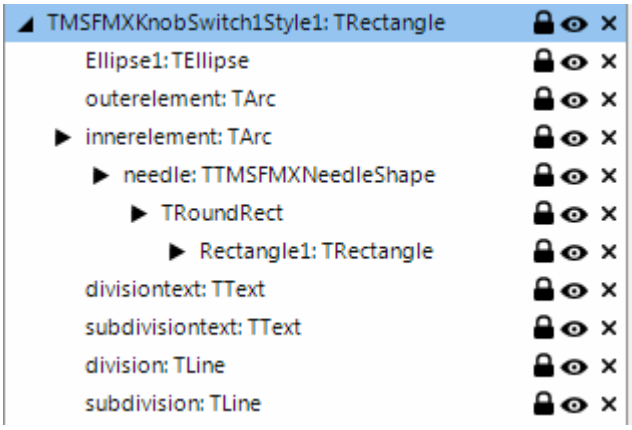
TTMSFMXRotarySwitch / TTMSFMXKnobSwitch

Two switch controls with a discrete but configurable number of positions are offered. The TTMSFMXRotarySwitch is a basic switch, the TTMSFMXKnobSwitch offers LEDs in the outer circle that indicate in what position the switch is.







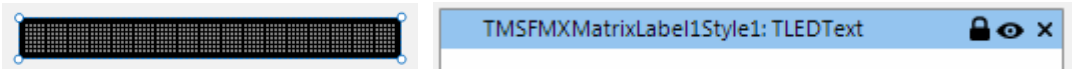


The values between with the switch control can select are set via the Positions collection. Each position has a Text property and a Tag property. For each position there is a value added on the switch. The position of the switch can be easily get or set via TTMSFMXRotarySwitch.Value. The first position value is 0, the second position 1, etc…

In the TTMSFMXKnobSwitch the division lines are replaced by ellipse shapes that have a default glow effect. The needle is replaced by a rectangle that represents the “knob”. The rectangle on the “knob” has a default glow effect that has the same color as the ellipse shapes. When changing the value, that actually is an integer between 0 and the count of the positions, the ellipse shapes light up from zero till the value of the position of the switch.



TTMSFMXMatrixLabel



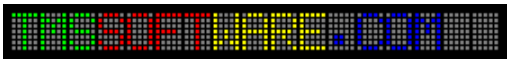
The matrix label has only one stylable element, the LEDText shape itself. This has properties to change the amount of LEDs, the size of the LEDs and the direction. The Text that is displayed inside the matrixlabel is set with the Text property in the Object Inspector.

The matrixlabel has the capability to autoscroll the text from right to left or vice versa. This is set with the ScrollDirection property. The text can be formatted as is, lowercase, uppercase or propercase. Propercase means that each word in the text starts with a capital letter.

Parts of the text can have different colors. This can be done by using a '%' character followed by a hex number from 0-F. This will change the color to a predefined set of colors. The color remains applied until a new color code is set in the text.

Example:

```
TMSFMXMatrixLabel1.Text := 'tms%Asoft%Cware%0.com'
```



Indexes of colors are:
%0: default color (green)
%1: black
%2: dark red
%3: dark green
%4: dark yellow
%5: dark blue
%6: dark fuchsia
%7: dark aqua
%8: gray
%9: silver
%A: red
%B: green
%C: yellow
%D: blue

%E: fuchsia
%F: aqua
Properties

- **AutoScroll**: Automatically scrolls the text from in the scrolldirection if true.
- **ScrollDirection**: The direction in which the text scrolls if the AutoScroll property is true.
- **Text**: The text of the label.
- **TimeInterval:** Sets the scroll speed.

Style element properties

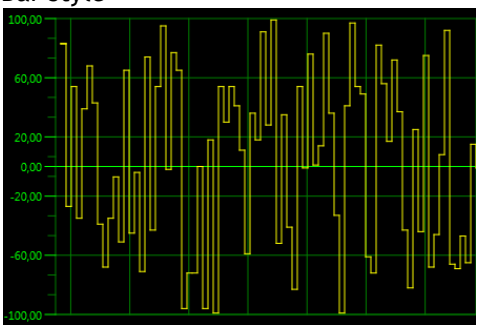
- **LEDs**: The number of visible matrix LEDs.
- **LEDStyle**: The size of the matrix LEDs : ls14x20, ls19x27, ls9x13.
- **LEDsVisible**: Shows / hides the inactive LEDs in the matrix.
- **Margin**: the amount of spacing between the top / bottom of the label and the LEDs.
- **Orientation**: LEDs can be displayed horizontal / vertical.
- **Spacing**: The amount of spacing between the LEDs
- **Text**: The text of the label. The text set in the StyleBook is overriden by the property at component level.
- **TextColor**: The default color of the text.
- **TextStyle**: The automatic casing that will be applied: uppercase, lowercase, propercase or as is.

TTMSFMXScope

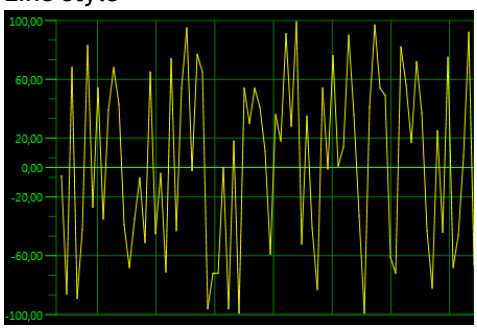
The TTMSFMXScope has the capability of displaying values of multiple channels with a certain interval and frequency. The scope cannot be styled as the scope has no default style implemented.

The scope has a channels collection property that can be used to add channels. Each channel has a Color, Font, ShowValue, Style, ValueFormat, Visible and Width property. The Style property is used to switch between line and bar styles.

Bar style



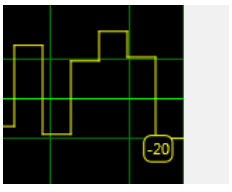
Line style



The scope is started with the Active property. With the AutoUpdate property set to true, the scope automatically scrolls from left to right or vice versa. The direction can be set with a separate property TTMSFMXScope.Direction. When the scope is scrolling, the OnNeedData event is called for each channel at each scroll step. The OnNeedData event has channel index parameter and a var Value parameter via which the value of the channel in the scope should be set.

The grid can be customized with the GridColor and GridLineWidth properties and the displayed values of the grid are customized via the Y-Axis class property. The Y-Axis has properties to format the divisions, subdivisions and set the position of the Y-Axis. The MinValue and MaxValue are set on the scope itself.

The scope can also display the last added value for each channel, styled with a font and the color of the channel. With the property ShowValue, the value is shown in a rounded rectangle, formatted with the ValueFormat property.



The value is displayed per channel, and is automatically positioned right or left in the grid, depending on the Direction property.

- **AnimateGrid / AutoUpdate**: Properties used to automatically scroll and animate the grid.
- **BaseColor**: The color used for the base center line.
- **Channels**: A collection which contains channels that are displayed inside the scope. Each Channel has a color, style, visible and width property. The Style defines the difference between line and bar.
- **Direction**: The scroll direction of the grid when autoupdating.
- **Frequency**: The frequency in pixels on which the data is drawn and updated.
- **GridColor / GridLineWidth**: The color and width of the grid lines.

- **MaxValue / MinValue**: The values that define the Y-Axis range and the area in which the channels are drawn.
- **YAxis**
  - **DivisionColor / DivisionTextColor / DivisionSize / DivisionFont**: The color of the main divisions between MinValue and MaxValue.
  - **SubDivisionColor / SubDivisionTextColor / SubDivisionSize / SubDivisionFont**: The color of the sub divisions between 2 division values.
  - **Divisions / SubDivisions**: The amount of values between MinValue and MaxValue.
  - **Format / FormatType**: The formatting of the values drawn in the Y-Axis.
  - **Position**: The position of the Y-Axis.
  - **Size**: The size of the Y-Axis.

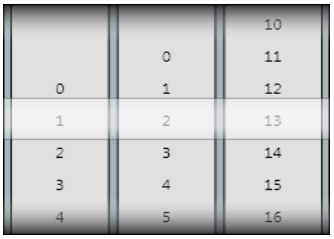
When AutoUpdate is set the false, the scope can still be programmatically updated. This is done via the method TTMSFMXScope.UpdateData(ChannelIndex, Value) and when the values for all channels have been set, TTMSFMXScope.AddData internally adds all channel data, updates the display and scrolls one step.

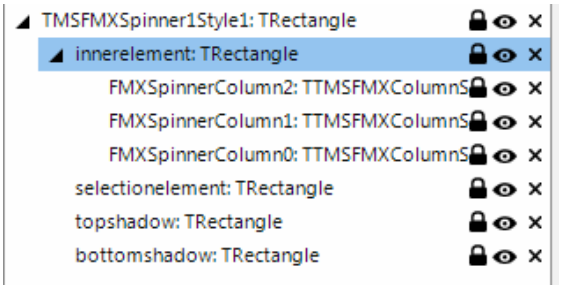
```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // initialization
  TMSFMXScope1.Channels.Clear;
  TMSFMXScope1.Channels.Add.Color := claRed;
  TMSFMXScope1.Channels.Add.Color := claYellow;
  TMSFMXScope1.Channels.Add.Color := claAqua;
  TMSFMXScope1.MinValue := 0;
  TMSFMXScope1.MaxValue := 30;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  // set values for each channel and step the scope
  TMSFMXScope1.UpdateData(0, Random(10));
  TMSFMXScope1.UpdateData(1, 10+  Random(10));
  TMSFMXScope1.UpdateData(2, 20 +  Random(10));
  TMSFMXScope1.AddData;
end;
```

TTMSFMXSpinner

TTMSFMXSpinner is a scrolling selector control as can be found in iPhone, iPad, iPod. It can be used to select a date or time but is versatile enough to select other types of data.





TTMSFMXSpinner style

The TTMSFMXSpinner style contains a minimum set of rectangles that define the layout. The topshadow and bottomshadow rectangles, the background and the selection rectangle. And the inner element rectangle which holds the columns.

TTMSFMXSpinner columns

The spinner has a Columns collection property, and when adding columns, they become available in the IDE style editor. Each column can be styled separately.

Each wheel of the spinner can be configured to scroll through a specific range of numbers, datetime values or custom values. The type of the range is set with RangeType property. For numbers, the range is set with RangeFrom and RangeTo properties.

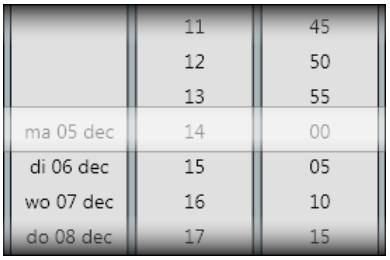
For a datetime range, the range is set with DateRangeFrom and DateRangeTo properties. The formatting of numbers and/or date time values displayed is controlled by DateTimeValueFormat or ValueFormat. For DateTimeValueFormat all formatting capabilities of the Delphi method FormatDateTime() are available. For the ValueFormat, all formatting capabilities of the Delphi method Format() are available. Each wheel has the option to make the wheel range cyclic. This is chosen by setting Cyclic = true. When this is true, the first range value is shown again immediately after the last range value and vice versa.

In the sample code snippet below, the spinner is configured to allow selecting a day and hour between now and 10 years.

```
TMSFMXSpinner1.Columns[0].RangeType := srtDateTime;
TMSFMXSpinner1.Columns[0].StepType := sstDay;
TMSFMXSpinner1.Columns[0].RangeFrom := Now;
TMSFMXSpinner1.Columns[0].RangeTo := Now + 365 * 10;
TMSFMXSpinner1.Columns[0].DateTimeValueFormat := 'DDD dd MMM';

TMSFMXSpinner1.Columns[1].RangeType := srtNumber;
TMSFMXSpinner1.Columns[1].StepType := sstNumber;
TMSFMXSpinner1.Columns[1].RangeFrom := 0;
TMSFMXSpinner1.Columns[1].RangeTo := 23;
TMSFMXSpinner1.Columns[1].ValueFormat := '%d';
TMSFMXSpinner1.Columns[1].Cyclic := true;
```

```
TMSFMXSpinner1.Columns[2].RangeType := srtNumber;
TMSFMXSpinner1.Columns[2].StepType := sstNumber;
TMSFMXSpinner1.Columns[2].Step := 5;
TMSFMXSpinner1.Columns[2].RangeFrom := 0;
TMSFMXSpinner1.Columns[2].RangeTo := 55;
TMSFMXSpinner1.Columns[2].ValueFormat := '%.2d';
TMSFMXSpinner1.Columns[2].Cyclic := true;
```

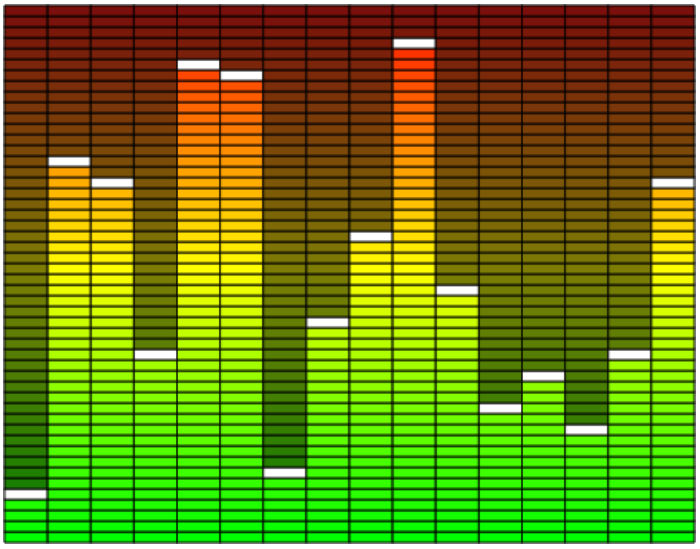


- **ColumnAppearance**
  - **AutoSize**: When true, size of the spinner columns is calculated automatically.
  - **Spacing**: Sets the spacing between the columns.
  - **TextSpacing**: Sets the spacing between the text drawn in the columns.

- **Columns**: A collection of spinner columns that can be configured in various ways.
  - **Cyclic**: When true, values are displayed as endless loop. The values are cyclic repeated when scrolling up or down.
  - **DateRangeFrom / DateRangeTo**: The range used when the rangetype is configured to used dates (srtDateTime). The step defines the amount of divisions between these 2 range properties.
  - **Font / FontAppearance**: The color and font of the text in a column.
  - **OnlyDate**: If the RangeType is set to use datetime values, the SelectedValue contains only the Date part.
  - **RangeFrom / RangeTo**: Defines the range of values used when setting the RangeType to srtNumber.
  - **SelectedValue**: Gets or sets the selected value. This property is used for both datetime range and the number range.
  - **Step**: The amount of steps between the start and end range in either number or datetime mode.
  - **StepType**: When not using srtNumber range type, the StepType can be set to increase per second, minute, hour, day, month or year.
  - **TextAlign**: Sets the alignment of the text within a column.
  - **ValueFormat**: Sets the formatting of the value when RangeType is srtNumber.
  - **Width**: Sets the width of the column if the Autosizing is false.

- **SmoothScrolling**: When dragging and holding the left mouse button the value changes. If smoothscrolling is false, the value is snapped inside the selected value area. If smoothscrolling is true, the value is only snapped when releasing the left mouse button.

TTMSFMXLEDMeter / TTMSFMXLEDScope



The TTMSFMXLEDScope is a collection of TTMSFMXLEDMeter components (channels). The scope displays a range of LED meters that have LED segments that be displayed in an active or inactive state. The LED meters are added through the Channels collection property and can be configured separately. Each LED channel has a Start-, Stop-, ActiveStart- and ActiveStopColor. The transition between these colors is automatically calculated for each LED segment in the meter.

The maximum value is set with the Steps property. This sets the number of small LED segments displayed. With the Value property, the number of LEDs highlighted with the Active color is set. Each channel has a peak value that is displayed as a white segment by default. The peak value is automatically raised to the maximum value of the channel. If the channel value lowers, the peak value remains on the highest value. The peak value is an optional indicator enabled with the ShowPeak property. The peak value can also be programmatically set via PeakValue.

A TTMSFMXLEDMeter can be treated as a single channel.

- **Channels**: a collection of channels that contain a LED meter shape.
    - **ActiveStartColor / ActiveStopColor / StartColor / StopColor**: Sets the color range for active and inactive states.
    - **PeakColor / PeakValue / ShowPeak**: Sets the color of the peak value, the peak value displayed and whether the peak value is visible.
    - **Steps**: The amount of segments between the StartColor and StopColor. The color transition is automatically calculated.

TTMSFMXLED / TTMSFMXLEDBar



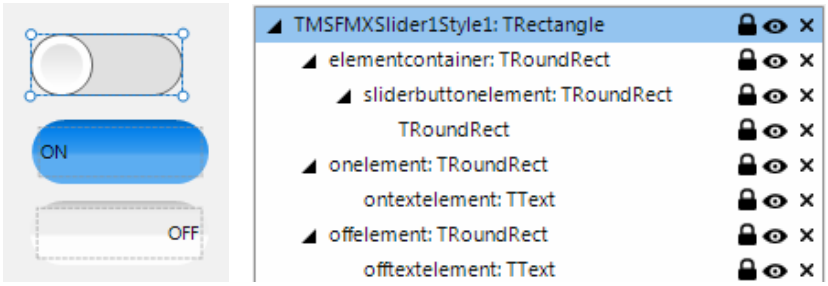
Similar to the TTMSFMXLEDScope, the TTMSFMXLEDBar is a collection of TTMSFMXLEDs that can be used separately. LEDs are added or removed via the LEDs collection or set with the Count property.

Just like with the scope, the bar is not stylable through the IDE style editor. The color of the LEDs can be changed per LED. Each LED has a base color which is used to automatically calculate the off and on color of the LED. With the state property the LED can be set to the on (true) or off (false) stat. In the active state, the LED uses the OnColor otherwise the OffColor. Note that setting the BaseColor at design time will automatically cause the LED to calculate a bright color for the on state and a darker color for the off state. If the automatic calculated OnColor or OffColor is not wanted, these properties can also be set directly.

The LEDBar has a Value property which automatically sets the state property of the LEDs to on for all LEDs that fall in range between 0 and the Value property.

- **Count**: The amount of LEDs displaying in the bar.
- **Leds**: A collection of LEDs used inside the bar. This way, each LED on the bar can be given a different color.
  - **BaseColor**: The color used to set the On- and OffColor.
  - **OnColor**: The color used when the LED is in the on state.
  - **OffColor**: The color used when the LED is in the off state.
  - **State**: The property that determines if the LED is on or off.
- **Spacing**: The spacing between the LEDs.

TTMSFMXSlider

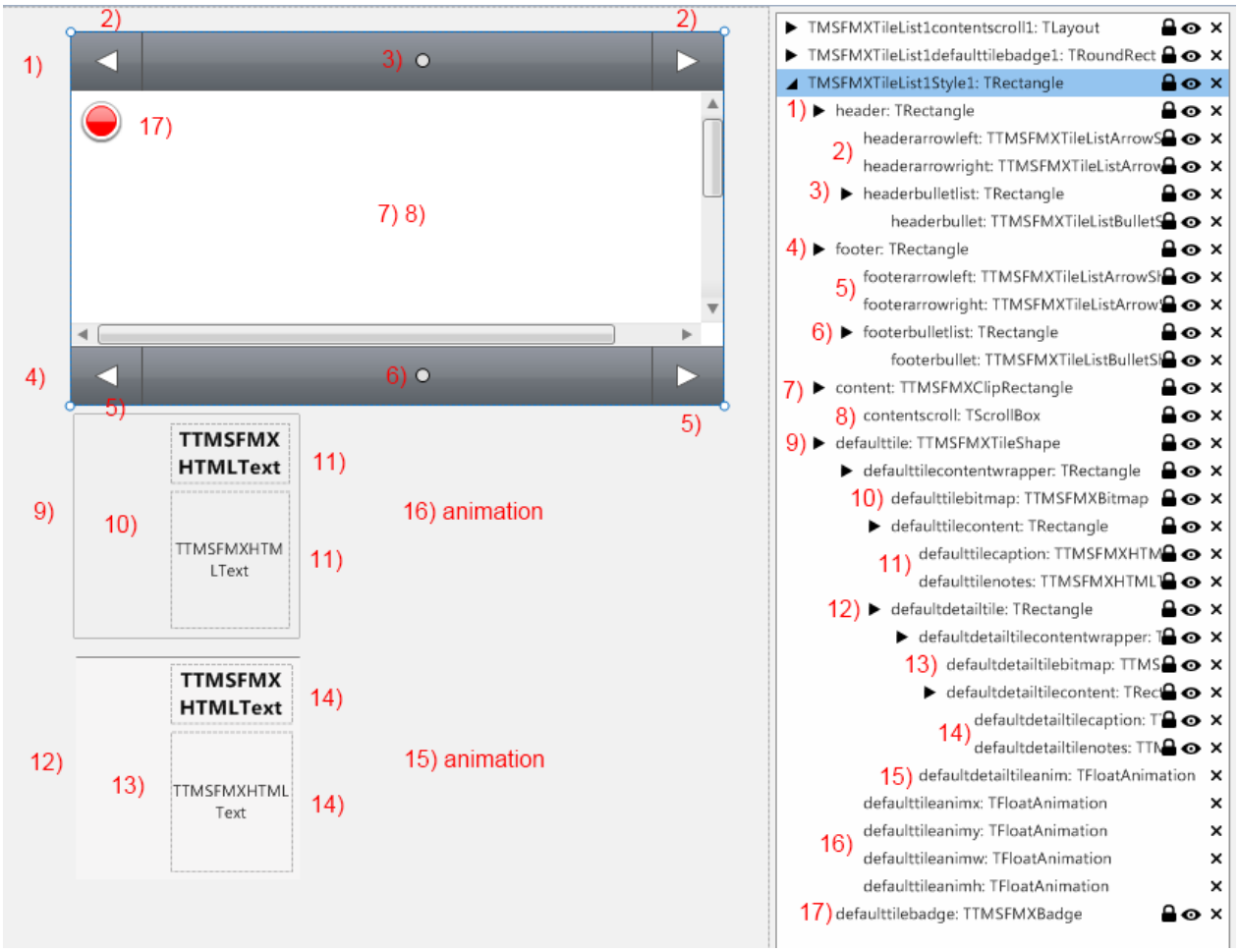


The slider consists of 4 elements that define the default layout: the elementcontainer which represents the background, the slider button element and the on and off element. When editing the slider via the IDE style editor, the on and off element are separated from the control to be visually styleable. After applying the style, the on and off elements are aligned inside the elementcontainer.

The slider has a State property that is used to switch between on and off. You can use the mouse to interact with the slider, clicking in the left or right part, or dragging the button, switches the state of the slider.

TTMSFMXTileList

Architecture



1) **Header**: The header of the tilelist that contains elements for navigation.
2) **Header Arrows**: The arrows for page navigation (left and right arrows)
3) **Header Bullets**: Bullets that indicate the number of pages and the current page index. Can be uses to navigate through pages.
4) **Footer**: The footer of the tilelist that contains elements for navigation.
5) **Footer Arrows**: The arrows for page navigation (left and right arrows)
6) **Footer Bullets**: Bullets that indicate the number of pages and the current page index. Bullets can be clicked to navigate through pages.
7) **Content**: The area where the tiles are displayed and where either page based or horizontal scrolling occurs.
8) **ContentScroll**: The scrolling area where the tiles are displayed and where the scrolling occurs. Depending on the scroll mode, either the tiles scroll by page or simply horizontally.
9) **DefaultTile**: Holds the default tile appearance and container for optional elements in tiles.
10) **DefaultTileBitmap**: The default tile bitmap that can be linked in various ways to an image. This also supports BitmapContainer.
11) **DefaultTileCaption / DefaultTileDescription**: Default elements in the tile used to display the caption and description of the tile.
12) **DefaultDetailTile**: The default tile appearance of the detail view (that appears when hovering over the main tile).

13) **DefaultDetailTileBitmap**: Similar to the DefaultTileBitmap but used for the for the detail view of the tile.
14) **DefaultDetailTileCaption / DefaultDetailTileDescription**: Similar to DefaultTileCaption, DefaultTileDescription but used for the detail view of the tile.
15) **DefaultDetailTileAnim**: The animation that is used to popup the detail tile over the main view of the tile.
16) **DefaultTileAnim***: Default animations to animate the tiles when reordering / dragging.
17) **DefaulTileBadge**: The default appearance of the tile badge that can be shown on top of the tile as an indicator.

Styling

With the FireMonkey design philosophy in mind, we have made the TileList completely styleable. When editing the custom or default style when right-clicking on the component (See: General Firemonkey component usage guidelines) the basic TileList layout is defined with several styleable elements. Elements can be removed, added and modified and updates are reflected in the component when applying the edited style.

Programmatically, almost every element is accessible with a function. When you want to style an element programmatically, you can use the appropriate function. Below is an overview of each element that is styleable at designtime and at runtime:

```
function GetHeader: TControl;
function GetFooter: TControl;
function GetContent: TControl;
function GetScrollContent: TControl;
function GetHeaderBullet: TControl;
function GetHeaderBulletList: TControl;
function GetFooterBulletList: TControl;
function GetFooterBullet: TControl;
function GetHeaderArrowLeft: TControl;
function GetHeaderArrowRight: TControl;
function GetFooterArrowLeft: TControl;
function GetFooterArrowRight: TControl;
function GetContentList: TControl;
function GetContentListAnim: TAnimation;
function GetContentListScroll: TControl;
function GetDefaultDetailTile(Source: TFMXObject): TControl;
function GetDefaultDetailTileCaption(Source: TFMXObject): TControl;
function GetDefaultDetailTileNotes(Source: TFMXObject): TControl;
function GetDefaultDetailTileBitmap(Source: TFMXObject): TControl;
function GetDefaultDetailTileContent(Source: TFMXObject): TControl;

function GetDefaultTileByName(AStyleName: String): TControl;
function GetDefaultTile: TControl;
function GetDefaultTileCaption(Source: TFMXObject): TControl;
function GetDefaultTileNotes(Source: TFMXObject): TControl;
function GetDefaultTileBitmap(Source: TFMXObject): TControl;
function GetDefaultTileBadge: TControl;
function GetDefaultTileContent(Source: TFMXObject): TControl;
```

Some functions require a parameter: Source: TFMXObject. The tilelist supports multiple custom default styles for the tile appearance. This is explained in a different chapter. When the parameter is nil, the default tile style is returned, else the custom default tile style can be found with the GetDefaultTileByName function that can be used as a parameter for the other functions.

These are the most important functions that return the element that is available in the default style or the style of the item when it is cloned. A complete list can be found when typing the name of the component and search through all the “Get*” functions and for the Tableview Item the functions that have “Shape” in the name. When an item is created, you will notice that each item has a Shape function. This shape function returns the cloned default tile from the default style. From each shape there are equivalents for the caption, bitmap, notes…. Below is an overview of the Shape* functions that can be uses to get access to the tile shape that has been created.

```
function ShapeDetail: TControl;
function ShapeDetailBitmap: TControl;
function ShapeDetailCaption: TControl;
function ShapeDetailNotes: TControl;
function ShapeDetailContent: TControl;
function ShapeBitmap: TControl;
function ShapeCaption: TControl;
function ShapeBadge: TControl;
function ShapeNotes: TControl;
function ShapeContent: TControl;
function ShapeContentWrapper: TControl;
function ShapeDetailAnimation: TAnimation;
function ShapeAnimationX: TAnimation;
function ShapeAnimationY: TAnimation;
function ShapeAnimationW: TAnimation;
function ShapeAnimationH: TAnimation;
```

Properties / Methods / Events

Below is a list of properties, methods and events in alphabetic order that expose the core functionality of the component and that need a short introduction before delving into the details of the component.

TTMSFMXTileList published properties

- **BitmapContainer**: The bitmapcontainer used in combination with the tiles collection to show a bitmap inside a tile.
- **Columns**: Defines the number of columns displayed on one page.
- **ColumnWidth**: Sets a specific columnwidth for a tile, instead of automatically calculating the width. When this property is different from 0 the Columns property is ignored.
- **Filtering**: Sets the type of filtering the tile list uses. The lfStart and lfRandom option creates a subset of tiles that matches the search string.
- **FilterMode**: Sets the filtering mode. This selects between filtering and searching in the tile list, showing / marking the tiles that that match the string.
- **HorizontalSpacing**: Horizontal spacing between tiles.
- **KeyBoardFilter**: Allows typing on the keyboard to filter / search in the list.
- **KeyBoardMode**: Switches the keyboard mode between page navigation and tile navigation.
- **LookupTime**: The time the lookup remains active in seconds. When the lookuptime has passed the lookup filter is reset to an empty string.
- **MultiSelect**: When true, selection of multiple tiles is enabled.
- **NavigationMode**: Selects between page mode and scroll mode.
- **PageHeight**: Sets the height of the page which can be used in combination with scroll navigation mode.
- **PageIndex**: The current selected page.
- **PageWidth**: The width of the page which can be used in combination with scroll navigation mode.
- **Reorder**: Enables reordering of tiles.

- **RowHeight**: Sets a specific rowheight for a tile, instead of automatically calculating the height. When this property is larger than 0 the rows property is ignored.
- **Rows**: Defines the number of rows available on one page.
- **SelectedTileIndex**: The last selected tile index.
- **Styles**: Collection of custom tile styles, cloned from the default tile style.
- **TileOptions**: Defines which elements are visible in a tile. This can improve overall performance since the unchecked elements are not cloned and thus not available in the tile.
- **Tiles**: The collection of tiles.
- **VerticalSpacing**: Vertical spacing between tiles.

Tile published properties in Tiles collection

- **Badge**: The text that is shown in the badge in the left top corner of the tile.
- **BitmapName**: The name of the bitmap that is used to show the bitmap inside the tile. The bitmap is only shown when loBitmap is set in TileOptions.
- **CanSelect**: Sets whether a tile can be selected.
- **Caption**: The caption of tile.
- **ColumnSpan**: Stretches the tile over the specified number of columns.
- **DataObject**: Reference to an object that contains additional information.
- **DataString**: An extra additional string object.
- **DetailBitmapName**: The name of the bitmap that is used to show the bitmap inside the detail tile.
- **DetailCaption**: The caption of the detail view of the tile.
- **DetailNotes**: The notes of the detail view of the tile.
- **DetailSizePercentage**: The percentage of the detail tile that is shown on top of the main tile. 100% means that the detail view of the tile covers the normal view entirely.
- **Empty**: Sets a tile as empty and counts it as a tile for calculating the position of the other tiles. An empty tile appears as a hole within other tiles on the page.
- **EnableDetail**: Enables showing the detail view of the tile on hovering.
- **Notes**: The notes of the tile.
- **ReadOnly**: Enables / Disables interaction with the tile.
- **RowSpan**: Stretches the tile over the specified number of rows.
- **Selected**: Set the tile in selected state when the CanSelect property is true.
- **Tag**: The tag property to identify the tile.
- **Transparent**: Hides the background of the tile.

Public procedures, functions, properties on TTMSFMXTileList level

- ```
  function GetLookupKey: String;
  ```
  Return the current lookupkey of the tile list when using keyboard mode.

- ```
  procedure ApplyFilter(AFilter: String);
  ```
  Applies a filter on the tile list, showing only the tiles with a caption that matches the filter

- ```
  function GetIndexAtPosition(ATile: TTMSFMXTile; X, Y: Single):
  Integer;
  ```
  Returns the index of the tile on X and Y position. This function is used to drag/drop tiles and uses the ATile parameter to exclude the index of the tile that is dragged.

- ```
  procedure ClearFilter;
  ```
  Clears the filtering.

- ```
  procedure ClearAllTiles(AShape: TTMSFMXTileShape);
  ```
  Clears all selected tile shapes except the shape that is passed through as a parameter.
  ```

- `procedure SelectAllTilesBetween(AStartTileIndex, AStopTileIndex: Integer);`
  Selects tiles between a given start and stop index. Not that it is required that multi selection is enabled.

- `procedure DeSelectAllTiles(ATile: TTMSFMXTile = nil);`
  Deselects all tiles except the tile that is passed as a parameter.

- `procedure SelectTiles(ArrTiles: array of integer);`
  Deselects all tiles that are passed as an array of tile indexes.

- `procedure BeginUpdate; override;`
  Method to use in combination with EndUpdate to limit the component recalculation and internal update calls to one single action.

- `procedure EndUpdate; override;`
  Method to use in combination with BeginUpdate to limit the component recalculation and internal update calls to one single action.

- `function FindTileIndexBy(ARow, AColumn: Integer): Integer;`
  Returns the tile index by a given Row and Column.

- `function FindNextTileIndexBy(ARow, AColumn, APageIndex: Integer): Integer;`
  Returns the next tile by a Row and Column on a specific page.

- `function FindPreviousTileIndexBy(ARow, AColumn, APageIndex: Integer): Integer;`
  Returns the previous tile by a Row and Column on a specific page.

- `function PageCount: Integer;`
  Returns the number of pages.

- `procedure NextPage;`
  Navigates to the next page.

- `procedure PreviousPage;`
  Navigates to the previous page.

- `procedure GoToPage(APageIndex: Integer);`
  Navigates to page number APageIndex. First page starts at index 0.

Public procedures, functions, properties on TTMSFMXTile level

- `function TileList: TTMSFMXTileList;`
  Returns the tilelist parent control of the tile.

- `function AllowDisplay: Boolean;`
  Function that returns whether a tile can be displayed or not (in combination with filtering)

- `function Shape: TTMSFMXTileShape;`
  Returns the shape that is created for the tile.

- `procedure UpdateTile;`
  Updates the tile display elements to match the settings in the tile collection item.

- `property TilePosition: TTilePosItem;`
  Returns the tile position inside the page, which contains information about the row and column and page index.

- `procedure ShowDetail;`
  Shows the detail view of the tile.

- `procedure HideDetail;`
  Hides the detail view of the tile.

- `property State: TTMSFMXTileState;`
  The State of the tile: normal, selected, down, hover.

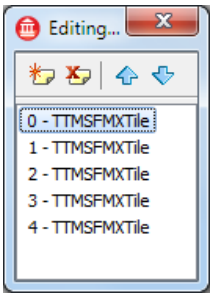
- `function GetBitmapContainer: TTMSFMXBitmapContainer;`
  Returns the BitmapContainer associated with the TileList when available.

Events published on TTMSFMXTileList level

- **OnBulletClick**: Event called when a page bullet in the header or footer is clicked.
- **OnLoadTile**: Event called when loading a tile for the first time. Additional settings for the tile can be dynamically applied from this event.
- **OnUnLoadTile**: Event called when unloading a tile when it is not necessary to display it anymore. (Browsing through the pages).
- **OnCustomizeTile**: Event called when a tile is created and customized. In this event, additional settings can be applied to the tile that are critical for a visual update when resizing, scrolling, etc…
- **OnCustomizeTileBadge**: Event called when a tile badge is created and customized.
- **OnCustomizeBullet**: Event called when a page bullet is created and customized.
- **OnNextPageClick**: Event called when clicking on the right arrow on the header or footer.
- **OnPreviousPageClick**: Event called when clicking on the left arrow on the header or footer.
- **OnTileClick**: Event called when clicking on a tile.
- **OnTileMouseDown**: Event called when the mouse button is down on a tile.
- **OnTileMouseUp**: Event called when the mouse button is released on a tile.
- **OnTileMouseMove**: Event called when the mouse is hovering over a tile.
- **OnTileMouseEnter**: Event called when the mouse is entering a tile.
- **OnTileMouseLeave**: Event called when the mouse is leaving a tile.
- **OnTileShowDetail**: Event called when the detail tile is shown.
- **OnTileHideDetail**: Event called when the detail tile is hiding.
- **OnFilter**: Event called when filtering the list (keyboard or programmatically).
- **OnFilterFinished**: Event called when filtering the list has finished (the lookuptime is exceeded).
- **OnTileStateChanged**: Event called when the state of a tile has changed (normal, hover, down, selected states).

Adding and removing tiles

Adding items can be done at designtime and at runtime. Click on the component to view the Tiles property. This is a default collection editor and can be used to add or remove items.



Items can also be added programatically:

```
var
  it: TTMSFMXTile;
  i: Integer;
begin
  TMSFMXTileList1.BeginUpdate;
  for I := 0 to 100 do
  begin
    it := TMSFMXTileList1.Tiles.Add;
    it.Caption := 'Item ' + inttostr(I);
    it.Notes := 'Hello World !';
  end;
  TMSFMXTileList1.EndUpdate;
```

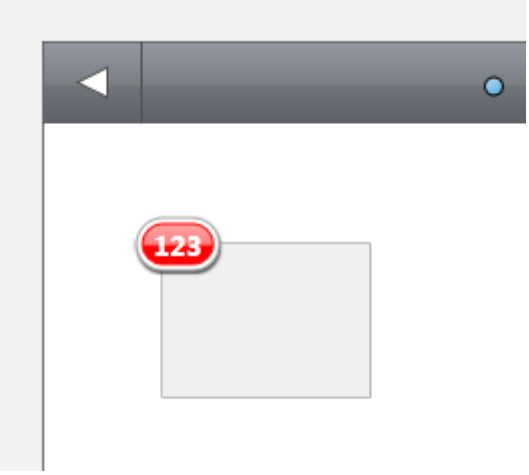
Whenever an item is added, the display list is normally rebuilt. Note that all calls to the TMSFMXTileList are embedded with BeginUpdate and EndUpdate calls here. With a BeginUpdate and EndUpdate call this update process to rebuild the display list is blocked and then executed once, i.e. when EndUpdate is called.
To remove an item programmatically, just call the item's destructor.

Badges

Each tile has a Badge property that is able to show additional information on top. The style for this badge can be modified in the default style of the tile list in the stylebook. To show a badge on a tile set the Badge property to the string of choice.

```
  TMSFMXTileList1.Margins.Left := 50;
  TMSFMXTileList1.Margins.Top := 50;
  TMSFMXTileList1.Margins.Right := 50;
  TMSFMXTileList1.Margins.Bottom := 50;

  with TMSFMXTileList1.Tiles.Add do
  begin
    Badge := '123';
  end;
```

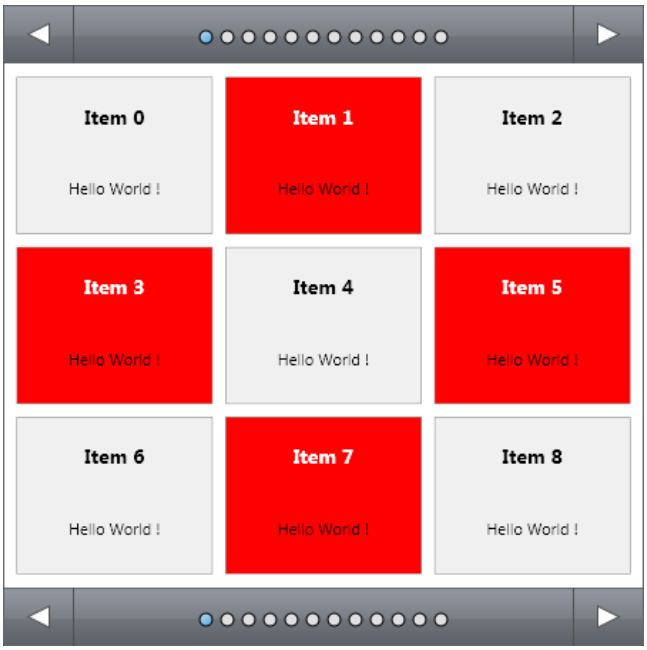


Tile Styles

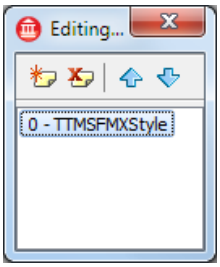
When opening the stylebook editor, you will notice the default tile style. This style is applied to all tiles that are added to the tilelist. Changing this style will be reflected in all the tiles. When you want to change the style of one or multiple tiles that must differ from the default style there are 2 options that can be used to customize the tile style.

1) Through the OnCustomizeTile event.
Below is a code sample that changes the default background color and caption color of all odd tiles (by tile index).

```
procedure TForm1.TMSFMXTileList1CustomizeTile(Sender: TObject;
  ATile: TTMSFMXTile; ATileShape: TControl);
var
  shp: TTMSFMXTileShape;
begin
  shp := ATile.Shape;
  if Assigned(shp) and Odd(ATile.Index) then
  begin
    shp.Fill.Color := claRed;
    (shp.ShapeCaption as TText).Fill.Color := claWhite;
  end;
end;
```

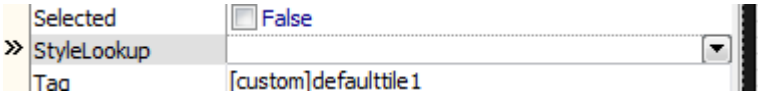


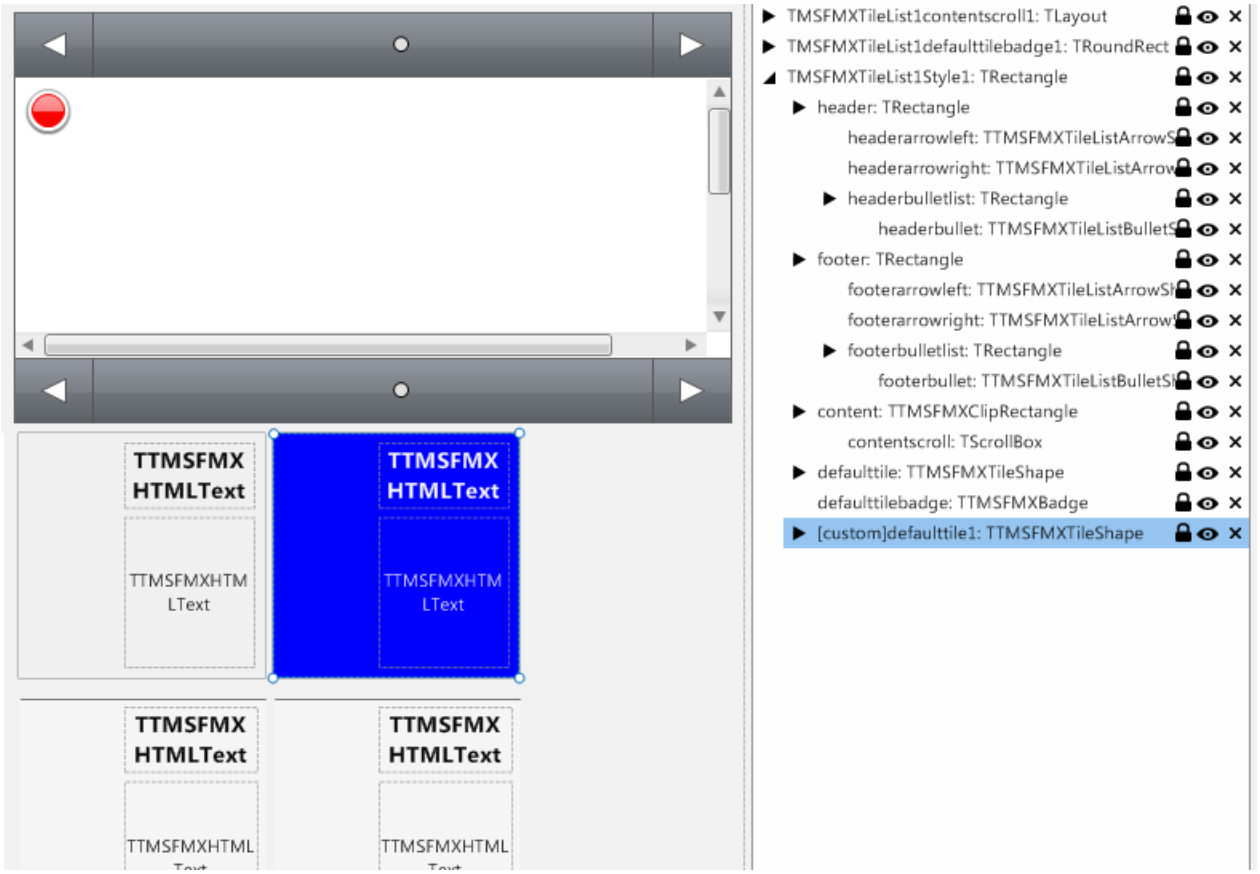
2) Through custom styles.
The tile list supports multiple custom styles that are cloned from the default style. With this technique the designing can be done through the stylebook editor instead of programatically. To create a new custom style, add a style item to the Styles collection of the tilelist.



The StyleName property of this style will automatically be named '[custom]defaulttile1'. You can change this before starting the stylebook editor. When making changes afterwards, the updated name will not be reflected in the stylebook! A good practice is to first add the number of custom styles needed in the list and then edit the style of the tilelist.

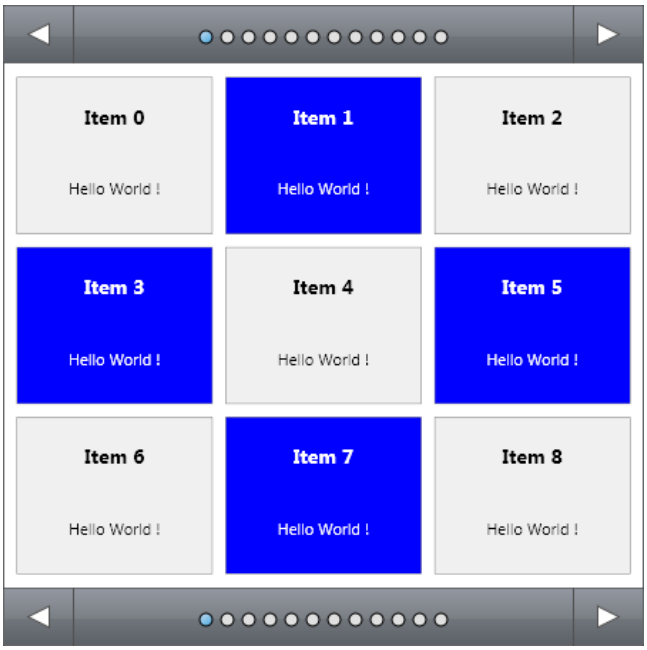
When right-clicking on the component and selecting "edit custom style…", a new custom tile style will be added next to the default tile style. You will notice that the new style is added to the treeview list on the right. When changing the appearance, the update will not be reflected immediately. To use this style inside the tilelist for specific tiles, you can select this style in the tiles collection editor. Clicking on the dropdown control for the property StyleLookup lists all available custom style to choose from.





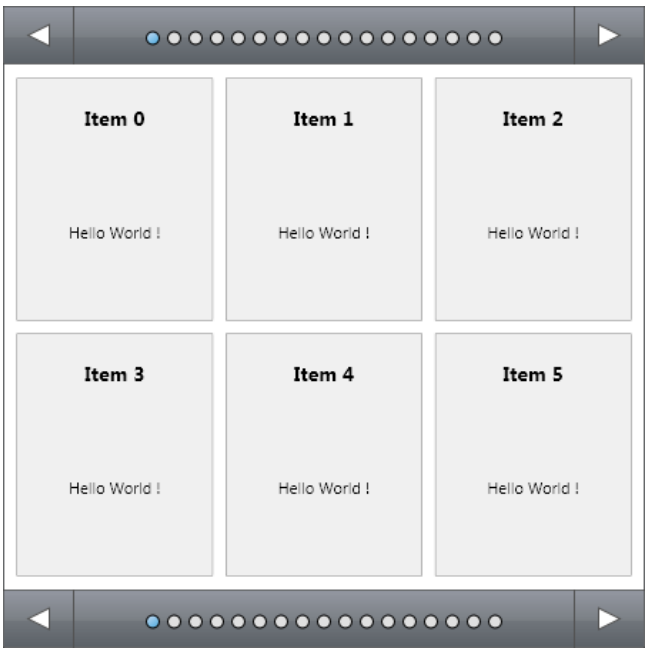
In this sample we have changed the background color of the custom tile style to blue and the caption and notes font color to white. Selecting this custom style or setting it programmatically after adding a tile in the tiles collection editor will use this style to create the tile shape.

```
var
  it: TTMSFMXTile;
  i: Integer;
begin
  TMSFMXTileList1.BeginUpdate;
  for I := 0 to 100 do
  begin
    it := TMSFMXTileList1.Tiles.Add;
    it.Caption := 'Item ' + inttostr(I);
    it.Notes := 'Hello World !';
    if Odd(I) then
      it.StyleLookup := '[custom]defaulttile1';
  end;
  TMSFMXTileList1.EndUpdate;
```



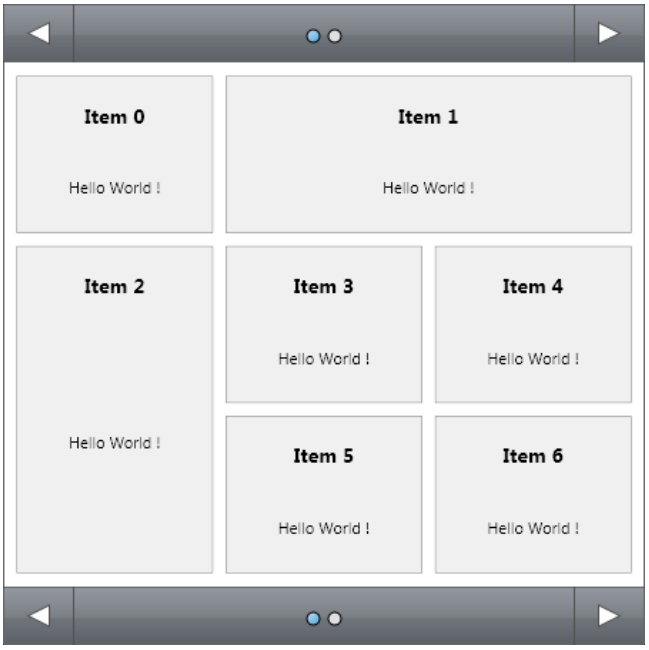
Columns and Rows

The tile list can contain multiple pages each consisting of multiple tiles organized in a grid that is defined with a columns and rows property. By default the tiles are organized in a 3 x 3 grid. Changing the grid layout to a 3 x 2 grid for example will give you the result below.



The tiles take an area of 1 cell by default. With the ColumnSpan and RowSpan properties on TTMSFMXTile level the tiles can be “stretched” over multiple cells. Below is a sample of some tiles that are reorganized by changing these properties.

```
var
  it: TTMSFMXTile;
  i: Integer;
begin
  TMSFMXTileList1.BeginUpdate;
  for I := 0 to 8 do
  begin
    it := TMSFMXTileList1.Tiles.Add;
    it.Caption := 'Item ' + inttostr(I);
    it.Notes := 'Hello World !';
    it.ColumnSpan := Random(4);
    it.RowSpan := Random(4);
  end;
  TMSFMXTileList1.EndUpdate;
```



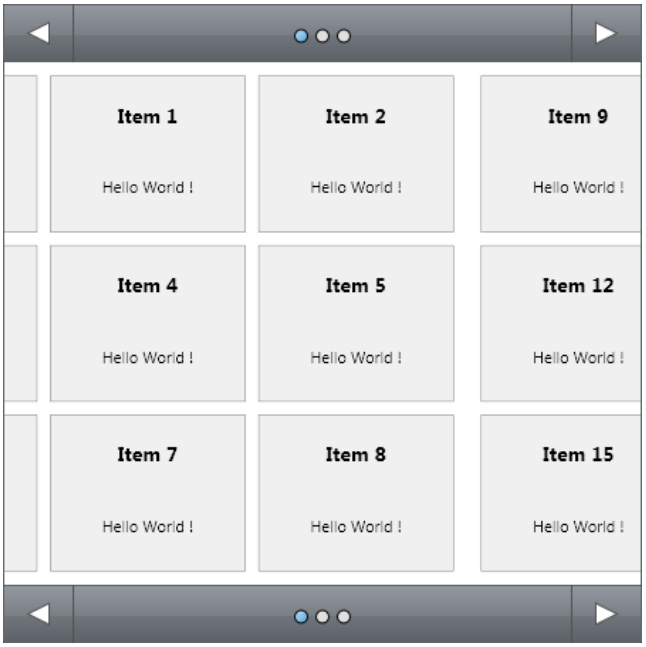
The Row, Column, ColumnSpan and RowSpan properties are automatically calculated to make sure no tiles overlap already “occupied” cells. If a tile does not have enough room on the page it supposed to be placed on, the tile is automatically placed on the next page.

Paging / Scrolling

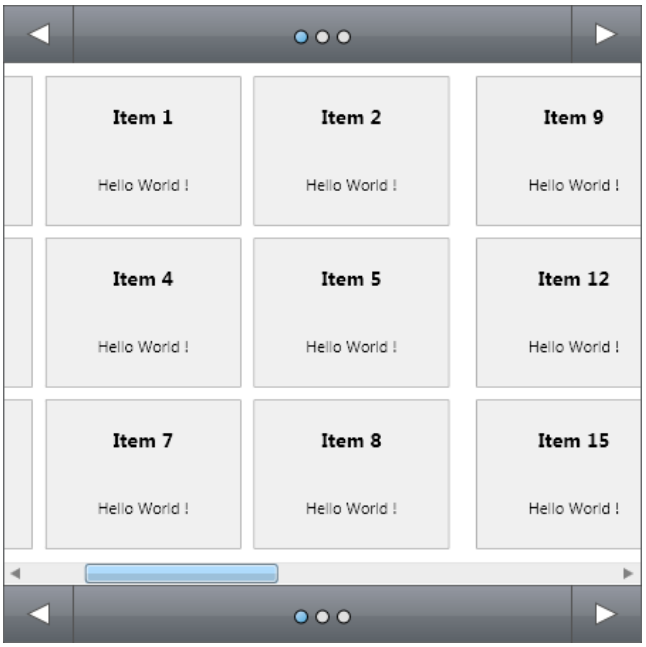
The tilelist supports two ways of displaying tiles. The tiles can either by organized in a page mode, displaying and loading only 3 pages (previous, current and next page), or in a scroll mode where all available tiles are loaded and displayed and a scrollbar is shown to scroll through the pages. To change between these 2 navigation techniques the property NavigationMode can be used. This property can be set to either nmPaging (Default) or nmScrolling.

Below is a sample that shows the difference between these 2 navigation modes:

Navigating through the list with nmPaging mode

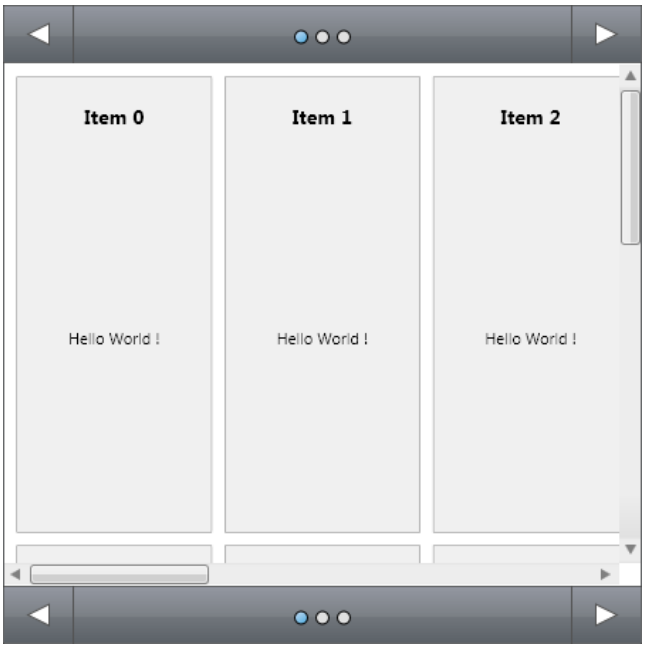


Navigating through the list with nmScrolling mode



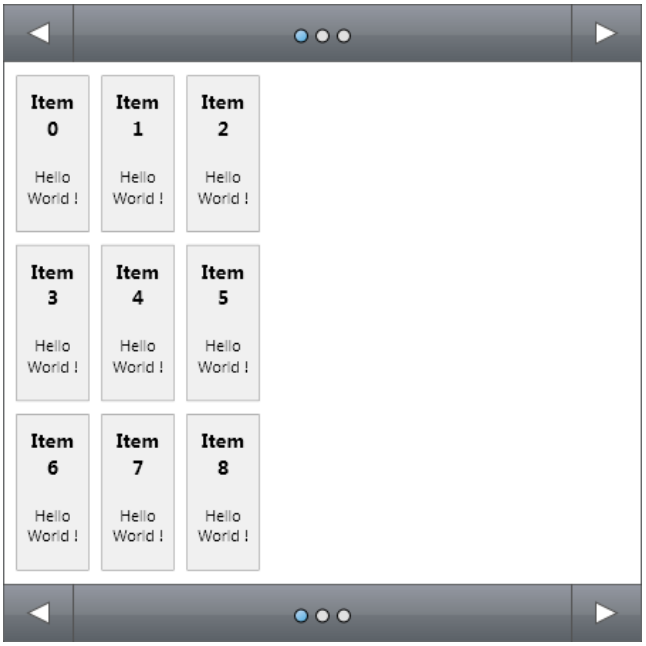
PageSize

The page dimensions are limited to the width and height of the content inside the tilelist by default. With the PageWidth and PageHeight properties, these dimensions can be modified. When the tile dimensions exceed the content list dimensions the vertical / horizontal scrollbars in the scrollbox will automatically appear when in nmScrolling navigation mode. Below is a sample with a pageheight that is higher than the side of the tile content list.



ColumnWidth / RowHeight

The tiles can also be distributed along a customized columnwidth / rowheight. These properties are available on tilelist level and are 0 by default. When 0, the width and height are automatically calculated based on the PageWidth / PageHeight /Columns and Rows property. Below is a sample when ColumnWidth is set to 50.



Filtering / Searching / Lookup

The tile list supports filtering, searching and looking up tiles programmatically as well as with the keyboard. Clicking on the list to set the focus and typing on the keyboard automatically filters the list based on the typed text. The text that is typed can be retrieved by using the GetLookupKey function. With the lookuptime property the time is set when the LookupKey is reset.

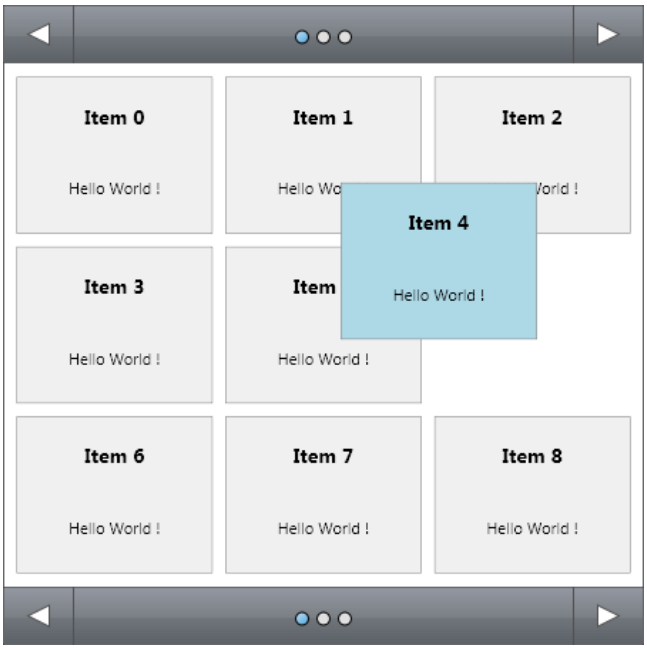
Depending on the Filtering, FilterMode properties the list either displays a subset of tiles or selects the tile that matches the lookup string.

Keyboard navigation

The tile list also supports navigation with the keyboard. By default the left, right, up and down arrow keys navigate through pages as if you are clicking on the page bullets. With the KeyBoardMode property this navigation can be changed to navigate within a page through the tiles. The arrow keys then select the next tile. If the tile is placed on the next page, the list will automatically navigate to the next page.

Reordering tiles

By default reordering tiles is enabled. When a tile is selected, the tile can be dragged on to a new location. To do this, first select the tile, then click and drag the selected tile to move it to a new position in the page. The reordering is done automatically when dragging the tile, proposing the closest new location at mouse coordinates for the tile to be dropped. Below is a screenshot of reordering in action:



When the tile is released, it will automatically animate to the new position.

Performance

A default TileList has some performance related options already preset to optimal values for the most typical use of the TileList. The amount of loaded pages is internally hardcoded to 3 (in nmPaging mode), which will only load and display the items that match the requirements to be displayed in one of the three pages. When navigating, the next page is loaded while the previous

page is removed, freeing memory and removing unnessary tiles. More performance related settings can be found in the TileOptions property. This contains a set of options that can be used to decrease the time the TileList needs to build up the display tiles list.

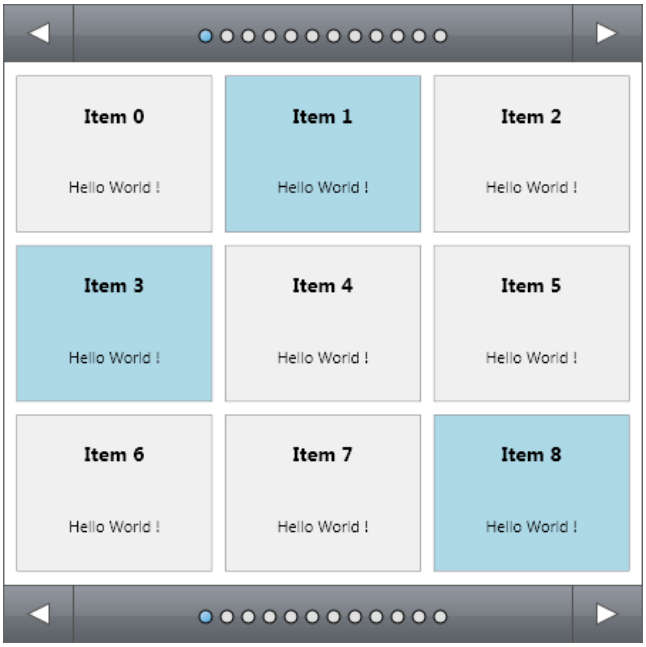
As explained in the Architecture chapter, the default tile contains various elements that are cloned when adding tiles. These setting under TileOptions affect what parts of a tile are cloned or not. When an element is not checked in the TileOptions, the element is not cloned when the display tile list is built and that increases the performance.

By default the TileOptions have unchecked loBitmap, loDetail and loDetailBitmap. This means that these elements will not be cloned and will not be available in the tile. You can further improve performance by unchecking elements that are not necessary for your application.

MultiSelect

The TileList supports multiple tile selection. Multiple tile selection can be enabled by setting the MultiSelect property to true. Clicking on a tile selects that tile and when you click on another tile, the previous tile is deselected. To select multiple tiles, you need to hold the CTRL or SHIFT key on the keyboard. With the CTRL key you are able to select tiles of choice and with the SHIFT key the list selects all tiles between the first and the last tile you have clicked.

MultiSelect can be done by keyboard alone, pressing the direction keys and holding the CTRL or SHIFT key at the same time. Below is a sample of multiple tiles selected in the list by holding the CTRL key and clicking several tiles.



Programmatically, in multiselect mode, the selected of an tile can be get or set with the tile Selected: Boolean property. In addition, three helper methods exist: SelectAllTilesBetween, SelectTiles, DeSelectAllTiles to programmatically select or unselect all Tiles at once.

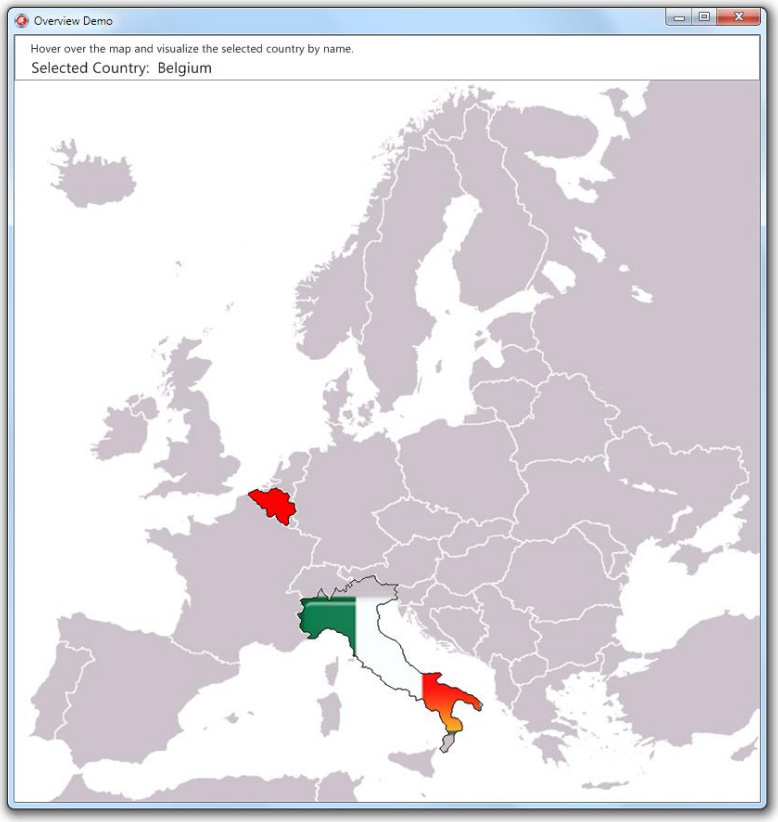
This sample code snippet programmatically selects items 1,3,5:

```
TMSFMXTileList1.MultiSelect := true;
TMSFMXTileList1.DeSelectAllTiles;
TMSFMXTileList1.SelectTiles([1, 3, 5]);
```

LiveBindings

The TTMSFMXTileList supports LiveBindings In the same way as the TTMSFMXTableView. More information can be found on the LiveBindings chapter at the TTMSFMXTableView component and demos, or at the TMS TileList LiveBindings Demo.

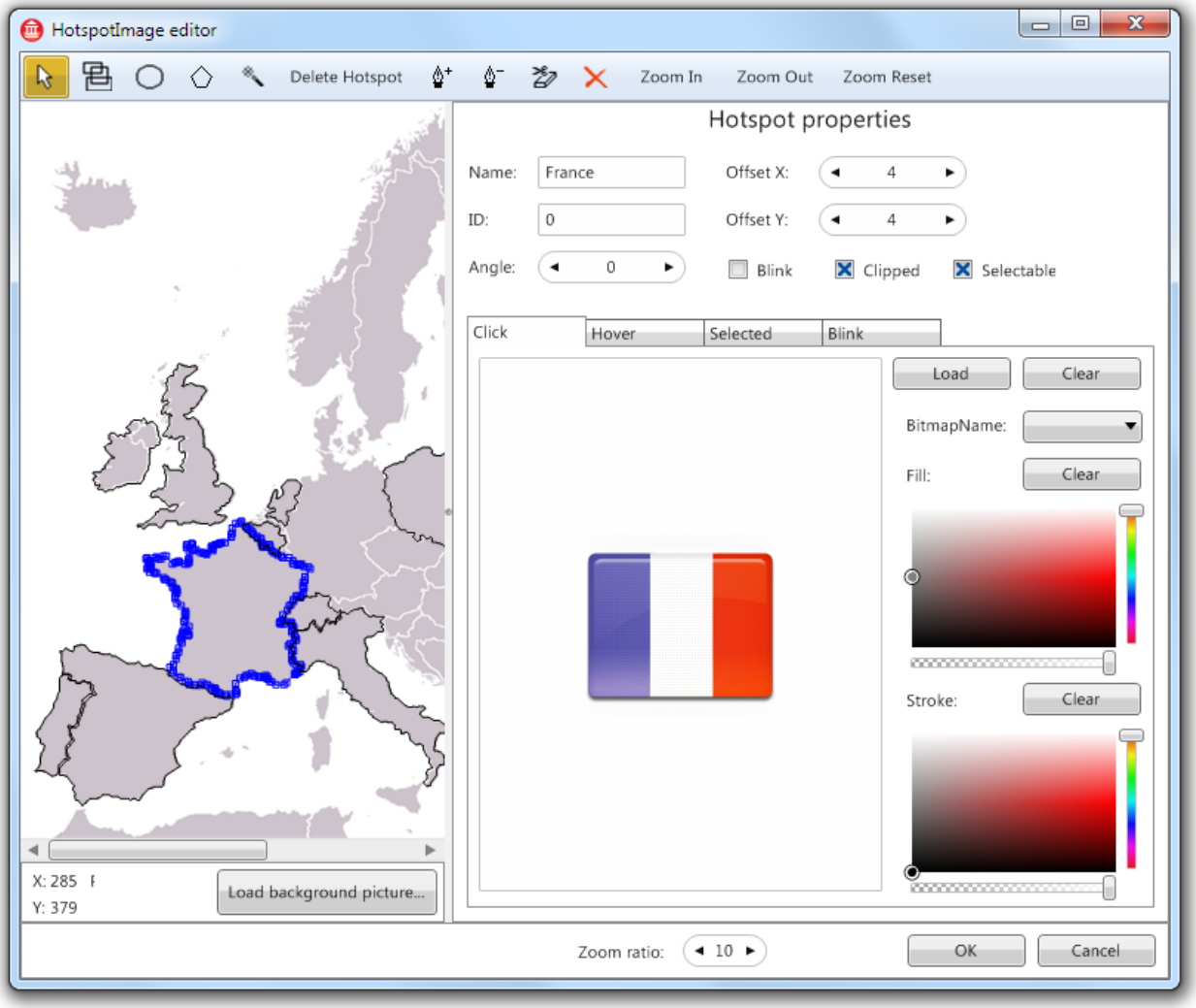
TTMSFMXHotSpotImage



The TTMSFMXHotSpotImage is a component that displays an image and that can have multiple areas on the images that are sensitive to mouse hover or mouse clicks. The areas on the images can be rectangular, circular or just irregular polygon shapes and for each area it can be choosen what color or what bitmap texture is displayed in the area on mouse hover or mouse click. With the hotspot editor it is easy to select or mark area's of the image you want to create a hotspot for. The hotspot editor is used at design-time but is also available for runtime use.

When dropping a TTMSFMXHotSpotImage on the form in the IDE, double-clicking the image starts the design-time hotspot editor. With the shape tools in the toolbar you can either create a rectangle, ellipse or polygon shape. The editor also has the capability of selecting an area with a magic wand tool. This selection is then converted to a polygon.

Selecting a hotspot enables the detail pane that shows the different states of the hotspot (Click, Hover, Selected or Blink). Each hotspot state supports a Fill, Stroke or Bitmap appearance. The bitmap can be loaded directly or through a bitmapcontainer that is assigned to the TTMSFMXHotSpotImage before starting the editor.

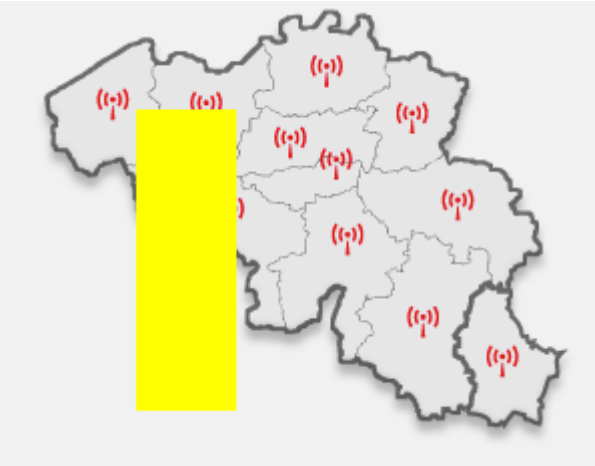


Adding a new hotspot

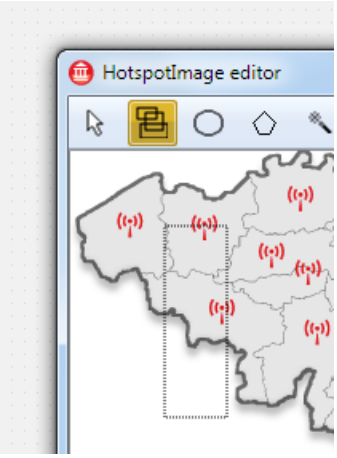
Adding a hotspot can be done programatically or at designtime.

To add a hotspot programmatically, use the following code:

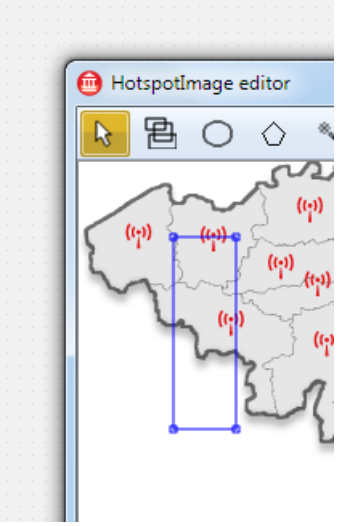
```
var
  hotspot: TTMSFMXHotSpot;
begin
  hotspot := TMSFMXHotSpotImage1.HotSpots.Add;
  hotspot.ShapeType := stRectangle;
  hotspot.X1 := 50;
  hotspot.Y1 := 50;
  hotspot.X2 := 100;
  hotspot.Y2 := 200;
```



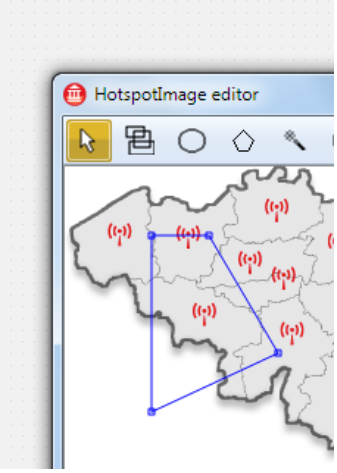
The same can be done at designtime. When double-clicking the component the editor starts. This will show the loaded image. To add a hotspot, select the tool of choice and click-drag and hold to create a new hotspot with the selected size and position:



When releasing the mouse, the hotspot will automatically be created and selected.



Properties can be changed in the right pane of the editor. Each hotspot can be further manipulated after creation, by dragging the handles to the position you want.



For each state the hotspot can be in, there is a stroke and fill. Only the color can be modified in the editor, the other brush related properties are accessible in the object inspector after closing the editor. Each state also supports displaying an image loaded directly or through a TTMSFMXBitmapContainer by defining just the image name for the hotspot.

Magic Wand Tool

In the editor, the magic wand tool automatically detects edges to convert to a hotspot of polygon type. Depending on the accuracy and tolerance the area will expand or contract. After selection of the magic wand tool the 2 sliders will become available to set these 2 additional parameters.



The magic wand tool is also exposed as method in the TTMSFMXBitmap component and can be used to automatically scan an area inside the loaded image. The function returns a polygon that contains a set of points that define the selected area. Below is a sample how to implement a Magic Wand tool at runtime, on a TTMSFMXBitmap.

```
var
   Poly: TPolyGon;

procedure TForm1.TMSFMXBitmap1MouseUp(Sender: TObject; Button:
TMouseButton;
  Shift: TShiftState; X, Y: Single);
begin
  poly := TMSFMXBitmap1.MagicWand(X, Y);
  // force the bitmap to repaint with the new created polyon
  TMSFMXBitmap1.Repaint;
end;

procedure TForm1.TMSFMXBitmap1Paint(Sender: TObject; Canvas: TCanvas;
  const ARect: TRectF);
begin
  Canvas.Stroke.Color := claWhite;
  Canvas.DrawPolygon(poly, 1);
end;
```

This code will select a part of the clicked image, turn it into a polygon, and paint it in a white stroked shape (see bottom part of the FireMonkey flame logo).



Saving and loading hotspots

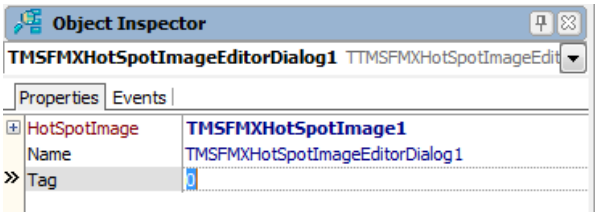
The TTMSFMXHotSpotImage supports saving and loading files that define the layout of a collection of hotspots. This can be achieved with the LoadFromFile and SaveToFile procedures. Images used inside the hotspots will also be loaded and hotspots will automatically be created when loading a file.

Compatibility

The TTMSFMXHotSpotImage component is designed with backwards compatibility in mind. Layouts which are saved with the VCL hotspotimage component are 100% compatible with the FireMonkey hotspotimage. With the FireMonkey hotspotimage, the colors of the different states are replaced by a fill. An additional stroke feature is added, which will load the default values when necessary.

TTMSFMXHotSpotEditorDialog

On the component palette you will also find a non-visual component TTMSFMXHotSpotEditorDialog next to the TTMSFMXHotSpotImage. This is a component that is able to show the hotspot editor at runtime. Simply drop the component on the form and select the TTMSFMXHotSpotImage in the dropdown of the HotSpotImage property in the Object Inspector.



Showing the dialog at runtime is a simple as calling the Execute function:

```
TMSFMXHotSpotImageEditorDialog1.Execute;
```

All changes at runtime will be reflected in the hotspotimage. The method TTMSFMXHotSpotImage.HotSpots.SaveToFile() could be used to save the runtime edited hotspots to file for example. Note that at runtime, the TTMSFMXHotSpotImageEditorDialog will work in Win32, Win64, Mac OS-X applications but is not supported for iOS, Android deployment.

TTMSFMXSpeedButton



The TTMSFMXSpeedButton descends from TSpeedButton and adds the capability to display an image. The speed button can be styled when right-clicking on the component and selecting either “edit custom style…” or “edit default style…”.

The speed button supports Staypressed functionality in combination with grouping. In other words, the speed button can act like a radiobutton / checkbox. To use this functionality, select the speedbuttons you wish to group and set the GroupName to the same string of choice for all selected components. Set the StaysPressed boolean property to true on all speedbuttons of the same group.

HTML is also supported as TTMSFMXHTMLText is used inside this button. More information can be found on the TMS Mini HTML Rendering Engine chapter.

TTMSFMXCalendar / TTMSFMXCalendarPicker



The TTMSFMXCalendar is a calendar which is able to display dates from a specific month in a various predefined date styles, such as weekend, current, normal and selected date styles. The important difference between the standard FireMonkey calendar and the TTMSFMXCalendar is the capability to perform multi and/or disjunct selection of dates and to indicate multiple events on specific dates.

Multiselect

The calendar can be used to select multiple dates, either a date range or a distinct selection of dates. To enable multiselect set Calendar.MultiSelect := true. In the calendar, a single click selects a day and dragging over the calendar, holding the left mouse button, selects a range of dates. With the keyboard, the shift and arrow keys can be used to select multiple dates. A combination between mouse and keyboard is also supported.

If multiselect is enabled, disjunct selection can be enabled with Calendar.DisjunctSelect := True. In this mode, the ctrl key is used to change the focused date. The date can be selected with the space key. All functionality with the shift date range selection remains active.

To get the selected dates, the calendar provides a collection of selected dates with Calendar.SelectedDays. A simple loop accessing the items of the collection returns the Date of that item giving you a set of selected days. In single selection mode, the Selected Date is returned with Calendar.SelectedDay.

Programmatic Selection

In single day selection mode a specific date can be set with

```
TMSFMXCalendar1.BeginUpdate;
TMSFMXCalendar1.SelectedDay := Now;
TMSFMXCalendar1.EndUpdate;
```

To clear selection in single day mode set

```
TMSFMXCalendar1.BeginUpdate;
TMSFMXCalendar1.SelectedDay := -1;
TMSFMXCalendar1.EndUpdate;
```

In multi / distinct selection mode, dates can be added to the SelectedDays array

```
TMSFMXCalendar1.BeginUpdate;
TMSFMXCalendar1.MultiSelect := true;
TMSFMXCalendar1.SelectedDays.Add.Date := Now;
TMSFMXCalendar1.SelectedDays.Add.Date := Now + 1;
TMSFMXCalendar1.SelectedDays.Add.Date := Now + 2;
TMSFMXCalendar1.EndUpdate;
```

To clear selection in multi day mode, clear the SelectedDays array.

```
TMSFMXCalendar1.BeginUpdate;
TMSFMXCalendar1.SelectedDays.Clear;
TMSFMXCalendar1.EndUpdate;
```

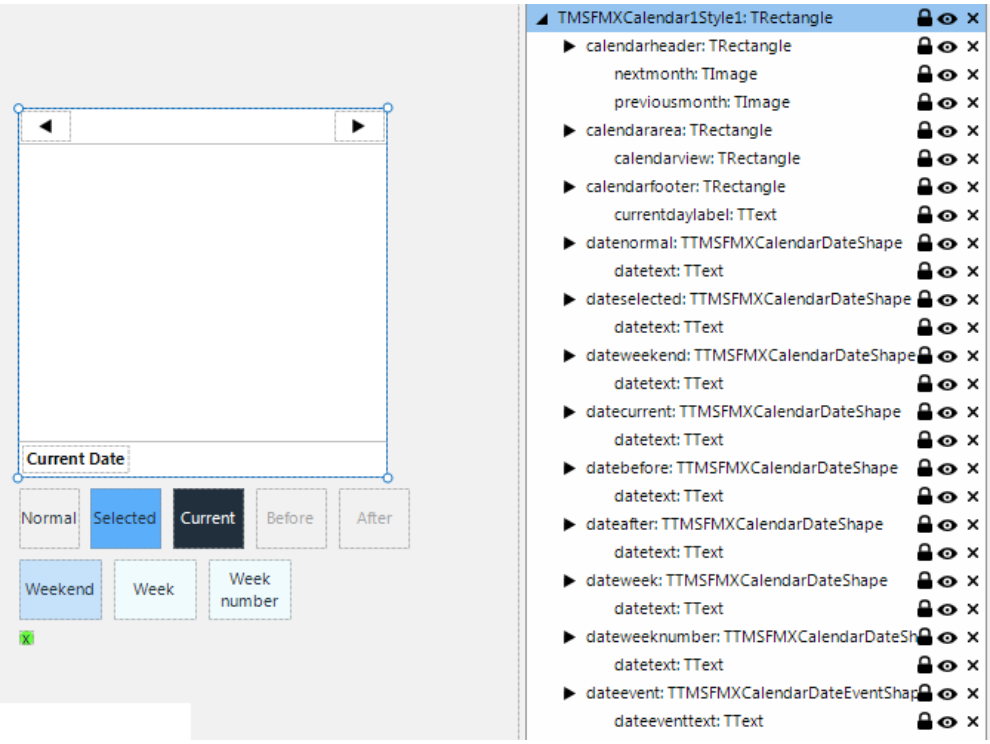
Events

The calendar exposes events for customization / interaction.

OnCurrentDayLabelClick: Event called when clicking on the current day in the footer.
OnCustomizeDay: Event called when adding the date shapes to the calendar, allows you to change the appearance, text and other important properties of a date shape. The object passed as a parameter is of type TTMSFMXCalendarDateShape.
OnCustomizeEvent: Event called when adding an event indicator shape to the calendar, allows you to change the appearance, text and other important properties of an event indicator shapes. The object passed as a parameter is of type TTMSFMXCalendarDateEventShape.
OnCustomizeWeekDay: Same event purpose as OnCustomizeDay but for the week days.
OnCustomizeWeekNumber: Same event purpose as OnCustomizeDay but for the week number.
OnDaySelect: Event called when selecting a day in the calendar.
OnDisjunctDaySelect: Event called when disjunct selection is performed.
OnEventClick: Event called when clicking on a calendar event.
OnEventShow: Event called when showing a calendar event.
OnEventHide: Event called when hiding a calendar event.
OnHasEvent: Event called per day to specify if that day has an event or not. If specified true, a default event is created for that day, which can be customized in the OnCustomizeEvent event.
OnMultiDaySelect: Event called when performing multi select.

Layout

Each date displayed in the calendar is styleable (chapter “FireMonkey Styles”), through the stylebook designer or the calendar events that are published. Each date shape applies the style that is defined in the default style of the calendar.



Events

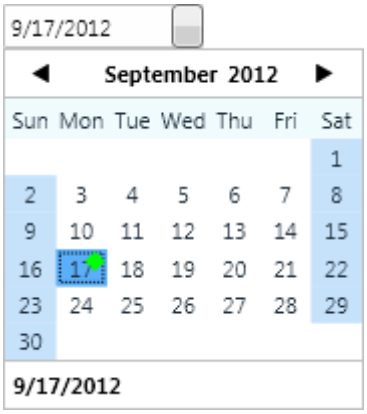
The calendar has the capability of displaying multiple events per day which show a popup on hovering. The calendar events are displayed with a customizeable indicator that can be set to a square, triangle, diamond or ellipse shape. To add an event to the calendar use

```
var
  evnt: TTMSFMXCalendarEvent;
begin
  evnt := TMSFMXCalendar1.Events.Add;
  evnt.Date := Now;
  evnt.Text := 'This is a new event!';
  evnt.Kind := ekDiamond;
```



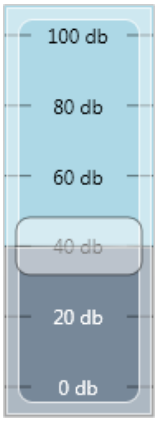
Calendar Picker

The calendar picker inherits from the TTMSFMXEdit component and displays a calendar in a dropdown control. The picker has the capability of displaying multiple selected dates as text with configurable delimiters.



The picker exposes a Calendar property that has all the properties and capabilities of a default calendar. To disable editing in the edit box of the picker the property CalendarPicker.ReadOnly := True can be used.

TTMSFMXTrackBar



The TTMSFMXTrackBar is a vertically or horizontally oriented trackbar with semi transparent thumb. The trackbar displays the range from minimum to position with a different fill than the range from position to maximum.

The TTMSFMXTrackBar allows to set a value between Minimum and Maximum. Tickmarks and value can be displayed at positions defined by TTMSFMXTrackBar.Step. The TTMSFMXTrackBar can be configured if and how the thumb should snap to the tickmarks. The formatting of the value is controlled with TTMSFMXTrackBar.ValueFormat and formatting that can be applied here is the same as available with the Delphi Format() function.

Snapping margins

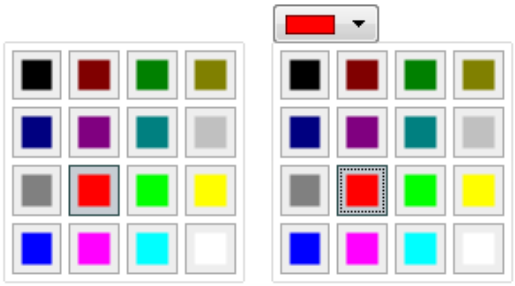
The property SnapMargin is used to snap the thumb to the values defined by the step property. Sample: When the SnapMargin is 10 and the step is 20, the thumb will immediately snap from 0 to 20, from 20 to 40. When the SnapMargin is 1 the thumb can be freely moved between 0 and 20 until the thumb reaches 19, then the thumb will automatically snap to the value 20.
When the SnapMargin is 0 there will be no snapping and the Thumb can be freely moved between minimum and maximum.

TTMSFMXButton



The TTMSFMXButton extends the TButton component and adds the ability to show an image and html formatted text. More information on the html formatted text can be found on the TMS Mini HTML Rendering Engine chapter.

TTMSFMXColorSelector / TTMSFMXColorPicker

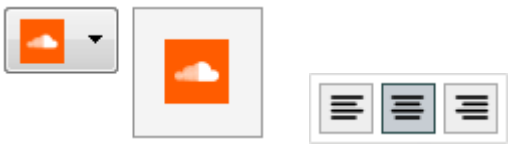


The TTMSFMXColorSelector and TTMSFMXColorPicker are components that are pre-configured, adding a standard set of colors to select from. Selecting a color is as easy as implementing the OnColorSelected event and/or programmatically retrieve the selected color with the TMSFMXColorSelector.SelectedColor or TMSFMXColorPicker.SelectedColor property. The picker variant displays the selector in a popup.

The TTMSFMXColorSelector and TTMSFMXColorPicker inherit from a base that allows a high level of customization. Each base supports an item collection that can be displayed in a column and row structure. Each item can be optionally hidden and/or disabled, stretched over a column and / or row span and can also be optionally configured as a seperator. The TTMSFMXColorSelector component overrides and adds a Color property to the base collection item class.

The base selector and picker classes support custom drawing on three levels: the background, the content and the text. A sample can be found at the TTMSFMXBitmapSelector / TTMSFMXBitmapPicker chapter.

TTMSFMXBitmapSelector / TTMSFMXBitmapPicker



The TTMSFMXBitmapSelector and TTMSFMXBitmapPicker are component that support displaying a collection of images to select from either directly in a selector or through a popup in a picker variant. Selecting a bitmap is as easy as implementing the OnBitmapSelect event and/or programmatically retrieve the selected Bitmap with the TMSFMXColorSelector.SelectedBitmap / TMSFMXColorSelector.SelectedItemIndex or TMSFMXColorPicker.SelectedBitmap property. The picker variant displays the selector in a popup.

The TTMSFMXBitmapSelector and TTMSFMXBitmapPicker inherit from a base that allows a high level of customization. Each base supports an item collection that can be displayed in a column and row structure. Each item can be optionally hidden and/or disabled, stretched over a column and / or row span and can also be optionally configured as a seperator. The TTMSFMXBitmapSelector component overrides and adds a Bitmap property to the base collection item class.

The base selector and picker classes support custom drawing on three levels: the background, the content and the text. Below is a sample that demonstrates this.

```
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  TMSFMXBitmapSelector1.BeginUpdate;
  TMSFMXBitmapSelector1.Items.Clear;
  TMSFMXBitmapSelector1.Columns := 3;
  TMSFMXBitmapSelector1.Rows := 1;
  for I := 0 to 2 do
    TMSFMXBitmapSelector1.Items.Add;
  TMSFMXBitmapSelector1.EndUpdate;
end;

procedure TForm1.TMSFMXBitmapSelector1ItemAfterDrawContent(Sender: TObject;
  ACanvas: TCanvas; ARect: TRectF; AItemIndex: Integer);
var
  pt: TPathData;
begin
  case TMSFMXBitmapSelector1.Items[AItemIndex].State of
  isHover: InflateRect(ARect,-4, -4);
  isDown,isSelected:
    begin
      InflateRect(ARect,-4, -4);
      ACanvas.Stroke.Thickness := 2;
      ACanvas.Stroke.Color := claBlack;
    end;
  isNormal: InflateRect(ARect,-8, -8);
  end;

  ARect := RectF(Int(ARect.Left)+ 0.5, Int(ARect.Top) + 0.5,
Int(ARect.Right) +0.5, Int(ARect.Bottom) + 0.5);

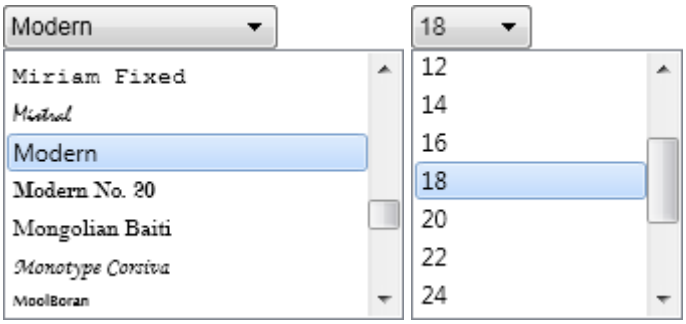
  case AItemIndex of
  0:
    begin
      ACanvas.Fill.Color := claBlue;
      ACanvas.FillEllipse(ARect,1);
      ACanvas.DrawEllipse(ARect,1);
    end;
  1:
    begin
      ACanvas.Fill.Color := claGreen;
      ACanvas.FillRect(ARect,0,0,AllCorners,1);
      ACanvas.DrawRect(ARect,0,0,AllCorners,1);
    end;
  2:
    begin
      pt := TPathData.Create;
      pt.MoveTo(PointF(ARect.Left + ARect.Width / 2, ARect.Top));
      pt.LineTo(PointF(ARect.Left + ARect.Width , ARect.Bottom));
      pt.LineTo(PointF(ARect.Left , ARect.Bottom));
      pt.ClosePath;

      ACanvas.Fill.Color := claRed;
      ACanvas.FillPath(pt,1);
      ACanvas.DrawPath(pt,1);
```

```
      pt.Free;
    end;
  end;
end;
```

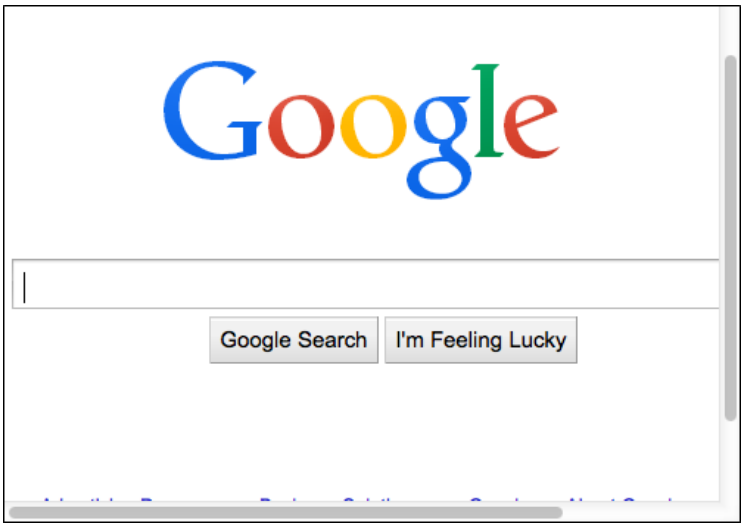


TTMSFMXFontNamePicker / TTMSFMXFontSizePicker



The TTMSFMXFontNamePicker and TTMSFMXFontSizePicker are components that are pre-configured, adding a standard set of font names and font sizes to select from. Selecting a font name / font size is as easy as implementing the OnFontNameSelected / OnFontSizeSelected event and/or programmatically retrieve the selected font name / Font size with the TMSFMXFontNamePicker.SelectedFontName or TMSFMXFontSizePicker.SelectedFontSize property.

TTMSFMXWebBrowser / TTMSFMXWebBrowserPopup



The TTMSFMXWebBrowser component is a wrapper around the native webbrowser component on Windows, iOS, Mac OS-X and Android. The TTMSFMXWebBrowser supports navigating to an URL, navigating forward, back in history, executing javascript, loading a file and displaying a HTML string. The TTMSFMXWebBrowserPopup can be used to display the TTMSFMXWebBrowser component modally inside a separate form or dialog depending on the target platform.

Loading a basic HTML string

```
TMSFMXWebBrowser1.LoadHTML('<body><h1>This is a <span style="background-
color:red;">HTML</span> string</h1></body>');
```



Loading a file

In this sample we load a file directly inside the TTMSFMXWebBrowser component The file that we load is a html page with the following content:

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

The code to load the file is

```
TMSFMXWebBrowser1.LoadFile(ExtractFilePath(ParamStr(0)) + 'test.html');
```



Executing Javascript

In this sample we load a html file directly inside the TTMSFMXWebBrowser, similar to the previous sample. The difference is that this file contains javascript to display the current date inside a label. The javascript function is called “DisplayDate()” and is called through a button click outside the TTMSFMXWebBrowser component. The html that is used in this sample is displayed below.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First JavaScript</h1>

<p>Click Date to display current day, date, and time.</p>

<p id="demo"></p>

<script>
function displayDate() {
   document.getElementById("demo").innerHTML = Date();
}
</script>

</body>
</html>
```

The code used to display the date from a button click:

```
TMSFMXWebBrowser1.ExecuteJavascript('displayDate();');
```

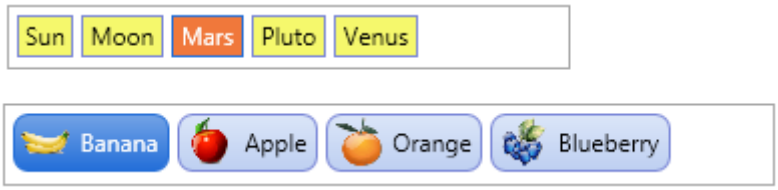


TTMSFMXSignatureCapture



Signature drawing component with import/export capabilities. Draw signature on desktop or mobile devices and export them to a stream, signature file or image. Compatible with VCL exported signature files. Optional clear button and text.

TTMSFMXListEditor



Architecture

TTMSFMXListEditor is an edit control to edit a list of values in a flexible way similar to the Microsoft Outlook or iOS email address input. It consists of a collection of items that can be edited, added, deleted via the control. Items are displayed in the control as clickable rectangular areas with an appearance that is controlled by the property TTMSFMXListEditor.ItemAppearance. In addition to text, each item can optionally also display an image before and/or after the text. The images can be clicked to perform further actions on.

Appearance

The appearance of the TTMSFMXListEditor is controlled by TTMSFMXListEditor.ItemAppearance. This property holds settings for normal state of items and for selected state. The settings include:

FillNormal: sets the background color of items in normal state
FontFillNormal: sets the text color of items in normal state
StrokeNormal : sets the color of the item border in normal state
RoundingNormal: sets the rectangle rounding of the item in normal state
FillSelected: sets the background color of items in selected state
FontFillSelected: sets the text color of items in selected state

StrokeSelected : sets the color of the item border in selected state
RoundingSelected: sets the rectangle rounding of the item in Selected state

Further, there is:
HorizontalSpacing : horizontal spacing in pixels between items in the list
VerticalSpacing : vertical spacing in pixels between items in the list
Note that the size of an item is determined by the text width & height (as well as optionally the width & height of a left and/or right image in the item). This means that to increase the height of an item for example, the font size shall be increased.

DefaultLeftImage, DefaultLeftImageName:
Sets the image or image name for the (optional) image on the left side of items. When DefaultLeftImage, DefaultLeftImageName is set, all new items get the image specified by DefaultLeftImage or DefaultLeftImageName.

DefaultRightImage, DefaultRightImageName:
Sets the image or image name for the (optional) image on the right side of items. When DefaultRightImage, DefaultRightImageName is set, all new items get the image specified by DefaultRightImage or DefaultRightImageName.

Note that the image on left side or right side can also be set per item via the item's LeftImage, LeftImageName and RightImage, RightImageName properties.

Note that in order to use DefaultLeftImageName or DefaultRightImageName, a TTMSFMXBitmapContainer must be connected to TTMSFMXListEditor.BitmapContainer. This is a container control that holds multiple images and these images can be accessed via a unique name identifier.

Items

TTMSFMXListEditor.Items is the collection that holds the items for the list. When the user adds or removes items, this is automatically reflected in the items collection. An item has following properties:

LeftImage, LeftImageName : sets the image to appear on the left side of the item
RightImage, RightImageName : sets the image to appear on the right side of the item
Tag : general purpose integer property
Text: holds the text of the item
Value: additional text property per item, available for storing extra information such as a hyperlink etc...

Adding items can be easily done via TTMSFMXListEditor.Items.Add.Text := 'New item' and deleting an item programmatically via TTMSFMXListEditor.Items.Delete(Index);

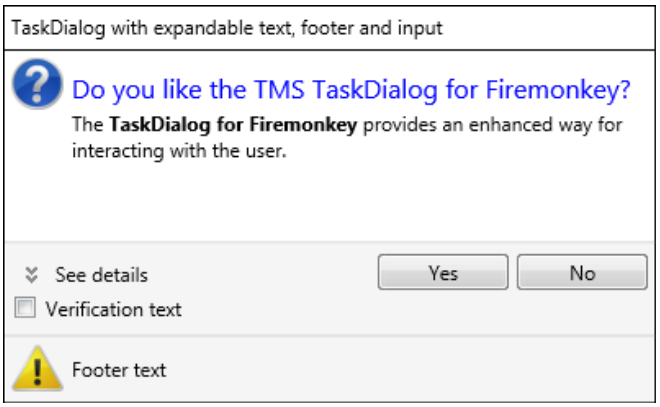
Events

In addition to the standard FireMonkey control events, TTMSFMXListEditor exposes some additional events relating to the process of editing items in the editor:

OnEditorApplyStyle: event triggered when the inplace editor control is about to be created and retrieved its style
OnEditorCreate: event triggered when the inplace editor is about to be created and allows to customize the editor class. The default editor class is TEdit
OnEditorGetSize : event triggered just before the inplace editor will be displayed in the control and allows to customize the size of the editor in the control

OnEditorGetText: allows to retrieve a text value for the value of the editor. When the inplace editor derives from TCustomEdit, the .Text property is automatically used but this event allows to use inplace editors that expose the value via another property than .Text for example.
OnEditorHide : event triggered when the inplace editor will be hidden
OnEditorShow : event triggered when the inplace editor will be displayed
OnEditorUpdate : event triggered when the value of the inplace editor has changed
OnItemCanDelete : event triggered when the user presses the DEL key for a selected item and allows to query for confirmation before the item is actually deleted
OnItemClick : event triggered when an item is clicked
OnItemDelete : event triggered when an item is deleted
OnItemInsert : event triggered when a new item is inserted via inplace editing
OnItemLeftImageClick : event triggered when the left image for an item is clicked
OnItemRightImageClick :event triggered when the right image for an item is clicked
OnItemUpdate : event triggered when the inplace editing stops and the value needs to be retrieved to update the item with.

TTMSFMXTaskDialog



A custom dialog component with expandable text, footer and input.
Additionally a progress bar, a list of radiobuttons, custom buttons or commandlinks can be displayed.

Properties

Bitmap: Set the image to display in the dialog
CollapsControlText: Text displayed when the ExpandedControlText is visible
CommandLinkBitmap: Set the image to display in a CommandLink
CommonButtons: Select the common dialog buttons to display
Content: HTML formatted text displayed in the dialog
CustomButtons: Set the names of the custom buttons to display
ExpandControlText: Text displayed when the ExpandeControlText is hidden
ExpandedText: HTML formatted text displayed when the ExpandControl is clicked
FooterBitmap: Set the image to display next to the footer text
FooterText: Text displayed in the dialog footer
InputSettings: Settings for input controls. Select which kind of control to display using the InputType property or select a custom control with the Control property
InstructionText: Text displayed as the dialog instruction
Options: Set if the CustomButtons are displayed as CommandLinks, if the Verify checkbox is checked and if a ProgressBar is displayed

RadioButtons: Set the list of radiobuttons to display
RadioButtonResult: Retrieve the index of the selected radiobutton
Title: Text displayed as the dialog title
VerificationText: Text to display with the Verify checkbox
VerifyResult: Indicates if the Verify checkbox is checked or unchecked

Methods
Show: Displays the TaskDialog
Show(ResultProc): Displays the TaskDialog and returns the index of the button that was clicked

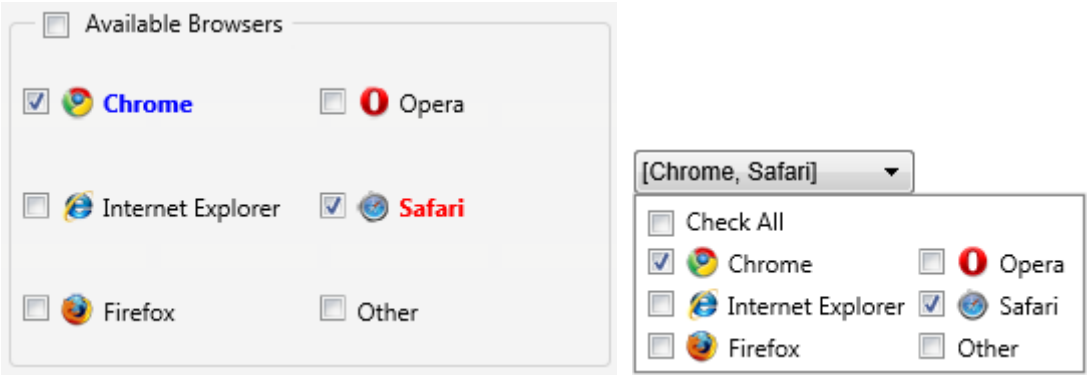
Sample code to display the TaskDialog and show which button was clicked in a listbox control:

```
var
  s: string;
begin
  TMSFMXTaskDialog1.Show(
  procedure(ButtonID: Integer)
  begin
    case ButtonID of
      mrOk: s := 'OK';
      mrCancel: s := 'Cancel';
    end;
    ListBox1.Items.Add(s + ' button clicked');
  end
  );
end;
```

Events

OnDialogAnchorClick: Event fired when an anchor element is clicked in the content text
OnDialogButtonClick: Event fired when a button from the CommonButton set or CustomButtons list is clicked
OnDialogClose: Event fired before the dialog is closed. Indicate if the dialog can be closed with the CanClose parameter
OnDialogInputGetText: Event fired when getting the text of a custom input control
OnDialogInputSetText: Event fired when setting the text of a custom input control
OnDialogProgress: Event fired when the progress bar is updated
OnDialogRadioClick: Event fired when a radiobutton is clicked
OnValidateInputText: Event fired when the input text is validated. Set if the input text is valid with the IsValid parameter

TTMSFMXCheckGroup / TTMSFMXRadioGroup
TTMSFMXCheckGroupPicker / TTMSFMXRadioGroupPicker



Components to display a group of checkbox or radiobutton controls with an optional image in a specified number of columns.

Properties

AutoSize: If set to true the control is automatically resized based on the number of items and their content, if set to false the items are automatically positioned and resized based on the size of the control.
Bitmap/BitmapName: Set the image to display next to the title text
BitmapHeight/BitmapWidth: Set the size of the images displayed next to the items
Bitmaps[Index]: Indexed property to retrieve a specific bitmap for an item
Columns: Set the number of columns to display the items in
GroupCheckBox: Set if a group checkbox is displayed next to the title text
GroupCheckBoxChecked: Set if the GroupCheckBox is checked
GroupCheckBoxType: Set which type of checkbox to display
- ctCheckAll: Check/uncheck all checkboxes (TTMSFMXCheckGroup only)
- ctDefault: Default checkbox behavior
- ctToggleEnabled: Toggle between enabled and disabled items

HTMLText[Index]: Indexed property to retrieve the text for specific item
IsChecked[Index]: Indexed property to retrieve the checked state for a specific item (TTMSFMXCheckGroup only)
ItemControls[Index]: Indexed property to retrieve a specific radiobutton or checkbox control
ItemIndex: The selected radiobutton index (TTMSFMXRadioGroup only)
Items: The collection of items to display
- Bitmap/BitmapName: Set the image to display next to the item
- Enabled: Set if the item is enabled or disabled
- Text: Set the (HTML formatted) text to display with the item

SelectedValue: Gets or sets the selected radiobutton based on the value (TTMSFMXRadioGroup only)
Text: HTML formatted title text
Value: Set which checkboxes are checked using a bitmask (TTMSFMXCheckGroup only)

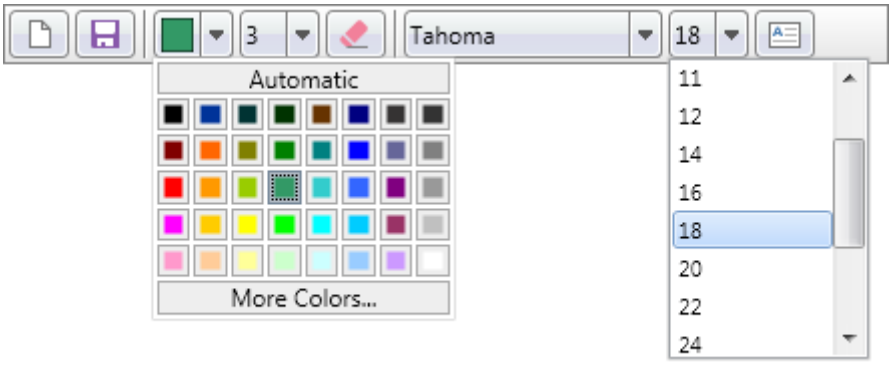
Methods
CheckAll: Sets state to checked for all checkbox items (TTMSFMXCheckGroup only)
UnCheckAll: Sets state to unchecked for all checkbox items (TTMSFMXCheckGroup only)

Events

OnCheckBoxChange: Event fired when a checkbox checked state is changed (TTMSFMXCheckGroup only)
OnCheckBoxClick: Event fired when a checkbox is clicked (TTMSFMXCheckGroup only)
OnGroupCheckBoxChange: Event fired when the GroupCheckbox state is changed
OnGroupCheckBoxClick: Event fired when the GroupCheckBox is clicked
OnRadioButtonChange: Event fired when the selected radiobutton is changed (TTMSFMXRadioGroup only)
OnRadioButtonClick: Event fired when a radiobutton is clicked (TTMSFMXRadioGroup only)

The TTMSFMXCheckGroupPicker/TTMSFMXRadioGroupPicker displays a CheckGroup/RadioGroup in a dropdown control.
The picker controls are similar in functionality and usage to the TTMSFMXCheckGroup and TTMSFMXRadioGroup. The selected item text is displayed in the selector area. The TTMSFMXCheckGroupPicker can display multiple selected items. The selected item(s) can be retrieved with the SelectedCheckBoxes/SelectedRadioButton property. The CloseOnSelection property of the TTMSFMXRadioGroupPicker can be set to true to automatically close the dropdown after a radiobutton has been selected. The OnCheckGroupSelected/OnRadioGroupSelected event is fired when checkbox or radiobutton state has changed.

TTMSFMXToolBar



The TTMSFMXToolBar is a component to display a group of toolbar buttons / pickers with optional separators. Each toolbar button is highly configurable and has the ability to show a dropdownbutton with a dropdowncontrol. There are also built-in font name, font size, bitmap and color pickers.

Set of components

- TTMSFMXToolBar
- TTMSFMXDockPanel
- TTMSFMXToolBarSeparator
- TTMSFMXToolBarButton
- TTMSFMXToolBarFontNamePicker
- TTMSFMXToolBarFontSizePicker
- TTMSFMXToolBarColorPicker

Properties

Appearance: The appearance of the toolbar which includes margins for automatic alignment of the controls inside the toolbar.
AutoAlign: Automatically aligns the controls inside the toolbar.
AutoSize: Automatically resizes the Toolbar according to the displayed buttons.
CustomOptionsMenu: A custom options menu, displayed when clicking the button at the right side of the toolbar. The options menu displays a list of controls that are available, and the controls can be hidden when clicking the appropriate item.
OptionsMenu: Configure the options menu at the right side of the toolbar.
State: The state of the toolbar. By default the state is esNormal, but when developing for mobile forms, the state can optionally be set to esLarge to allow larger buttons and sharper graphics.

Methods

AddControl(AControl: TControl; AIndex: Integer = -1);
Adds an existing control to the toolbar, optionally at a specified index.

AddControlClass(AControlClass: TControlClass; AIndex: Integer = -1): TControl;
Adds a new control based on the AControlClass parameter, optionally at a specified index.

AddButton(AWidth: Single = -1; AHeight: Single = -1; AResource: String = ''; AResourceLarge: String = ''; AText: String = ''; AIndex: Integer = -1): TTMSFMXToolBarButton;

Adds a new TTMSFMXToolBarButton with the ability to configure the button size, normal bitmap and large bitmap resources, text and position within the toolbar.

AddSeparator(AIndex: Integer = -1): TTMSFMXToolBarSeparator;
Adds a new separator to the toolbar.

AddFontNamePicker(AIndex: Integer = -1): TTMSFMXToolBarFontNamePicker;
Adds a new TTMSFMXToolBarFontNamePicker control, which inherits from TTMSFMXToolBarButton.

AddFontSizePicker(AIndex: Integer = -1): TTMSFMXToolBarFontSizePicker;
Adds a new TTMSFMXToolBarFontSizePicker control, which inherits from TTMSFMXToolBarButton.

AddColorPicker(AIndex: Integer = -1): TTMSFMXToolBarColorPicker;
Adds a new TTMSFMXToolBarColorPicker control, which inherits from TTMSFMXToolBarButton.

AddBitmapPicker(AIndex: Integer = -1): TTMSFMXToolBarBitmapPicker;
Adds a new TTMSFMXToolBarBitmapPicker control, which inherits from TTMSFMXToolBarButton.

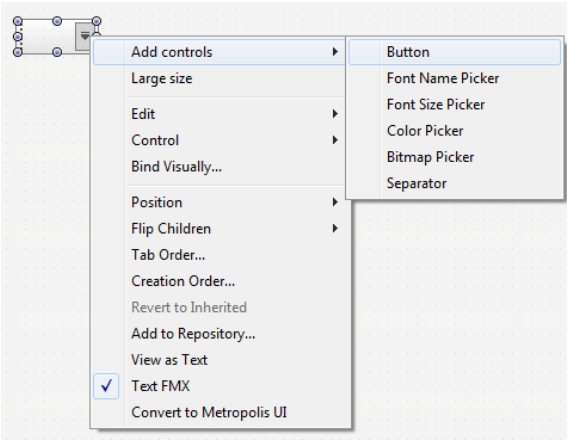
GetOptionsMenuButtonControl: TTMSFMXToolBarButton;
Returns the right-most options menu button for further customization.

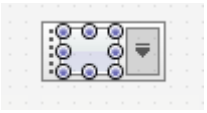
Events

OnOptionsMenuButtonClick: Event called when the menu button at the right side of the Toolbar is clicked.
OnOptionsMenuCustomize: Event called after the options menu is initialized and further customizations need to be applied.
OnOptionsMenuItemApplyStyle: Event called when the menu item style is applied.
OnOptionsMenuItemCanShow: Event called when showing
OnOptionsMenuItemClick: Event called when a menu item is clicked.
OnOptionsMenuItemCustomize: Event called when a menu item is initialized and further customization is necessary.

Adding new components at designtime

When dropping a TTMSFMXToolBar on the form, right-clicking it will give you a context menu with options to add controls. Adding a Button will create an instance of TTMSFMXToolBarButton and add it to the TTMSFMXToolBar. By default the AutoSize and AutoAlign is true which will align the button according to the properties set in the appearance and the width/height of the control.





The TTMSFMXToolBarButton can be further customized through the object inspector. The TTMSFMXToolBarButton has a few descendants that are listed in the beginning of this chapter, each inherit all properties from the TTMSFMXToolBarButton and already configure some properties to suit their purpose. The most important properties, methods and events are listed below.

Adding new components at runtime

For this sample we are taking the previous sample of adding a new TTMSFMXToolBarButton at designtime. The toolbar has a few helper methods of adding a new or existing control programmatically.

```
var
  b: TTMSFMXToolBarButton;
begin
  b := TMSFMXToolBar1.AddButton(100, 30);
  b.Text := 'Hello World !';
```



We can also add other non-built in type of controls, such as a TEdit.

```
var
  e: TEdit;
begin
  e := TMSFMXToolBar1.AddControlClass(TEdit) as TEdit;
  e.Text := 'Hello World !';
```



Toolbar button

Below are the most important properties, methods and events for the TTMSFMXToolBarButton.

Properties

Appearance: The appearance of the button, which includes fill and stroke for all states of the button including a optional transparent mode and the ability to change the corners and rounding.
AutoOptionsMenuText: The text that is displayed when clicking the options menu button in the toolbar.
Bitmap: The bitmap for normal state.
BitmapContainer: A container of bitmaps defined by a name property.
BitmapLarge: The bitmap for large state.
BitmapName: The name of the bitmap in normal state used in combination with the BitmapContainer.
BitmapNameLarge: The name of the bitmap in large state used in combination with the BitmapContainer.
DropDownControl: A reference to the control displayed inside the dropdown area.
DropDownHeight: The height of the dropdown area where the dropdown control will be displayed.

DropDownKind: The kind of dropdown button configured inside the toolbar button. When setting the DropDownKind to ddkDropDownButton a separate button is added to the toolbar button which takes care of displaying the dropdown. When specifying ddkDropDown, the whole toolbar button area will trigger a dropdown.
DropDownPosition: The position of the dropdown button.
DropDownWidth: The width of the dropdown area where the dropdown control will be displayed.
State: The state of the button, used to show the difference between normal and large states for desktop and mobile applications.

Methods

GetDropDownButtonControl: TTMSFMXToolBarDropDownButton;
Returns the internally created dropdown control button for further customization.

GetBitmapControl: TTMSFMXBitmap;
Returns the internally created instance of TTMSFMXBitmap used to display a bitmap inside the toolbar button.

GetTextControl: TTMSFMXHTMLText;
Returns the internally created instance of TTMSFMXHTMLText used to display the text inside the toolbar button.

DropDown;
Shows the dropdown area.

CloseDropDown;
Closes the dropdown area.

GetPopupControl: TPopup;
A reference to the popup control used to display the dropdown area.

DownState: Boolean
A special state that forces the downstate on the toolbar button.

PopupPlacement: TPlacement
The placement of the dropdown area. By default the dropdown area is shown with the direction set to bottom.

Normal State vs Large State

The button implements a state property, which is also available on the toolbar and dock panel. When setting the state property, the buttons are switched to a larger state, and will display a larger font size, larger size and larger bitmap. The bitmap is the most important because the bitmap will be loaded from the BitmapLarge properties. When you configure your application to include large states, you should also include a large state variant for the Bitmap and / or BitmapName properties.

Below is a sample that includes a bitmap for normal and large state.

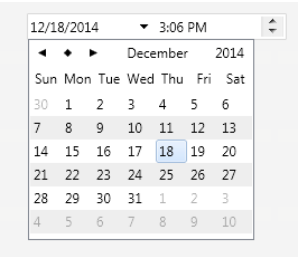
Normal state



Large state



TTMSFMXDateTimeEdit



The TTMSFMXDateTimeEdit combines the default TDateEdit and TTimeEdit component in FireMonkey and exposes properties, such as TimeFormat and DateFormat to easily configure both controls by displaying them as a single component.



Below is a sample that changes the format for Date and Time and sets the DateTime property

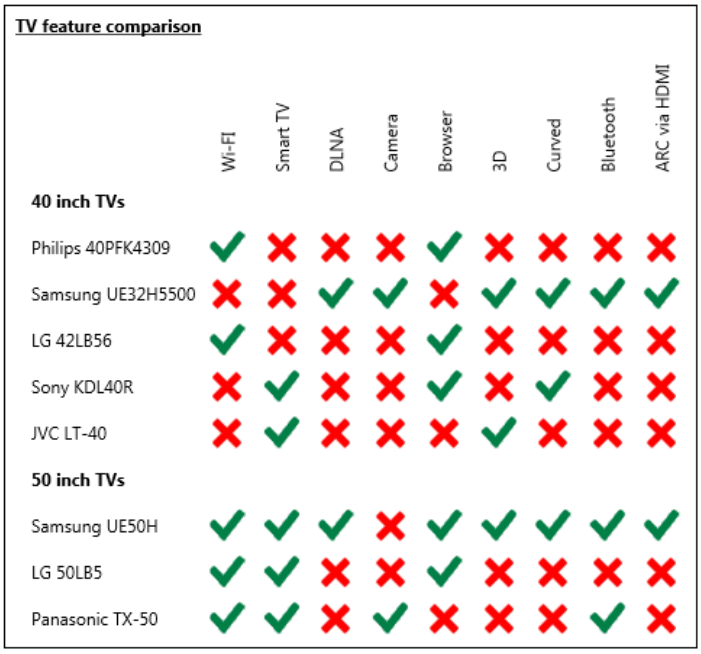
```
TMSFMXDateTimeEdit1.DateTime := EncodeDate(2014, 12, 10) + EncodeTime(14,
30, 30, 0);
TMSFMXDateTimeEdit1.TimeFormat := 'hh:nn:ss';
TMSFMXDateTimeEdit1.DateFormat := 'dddd mm, yyyy';
```

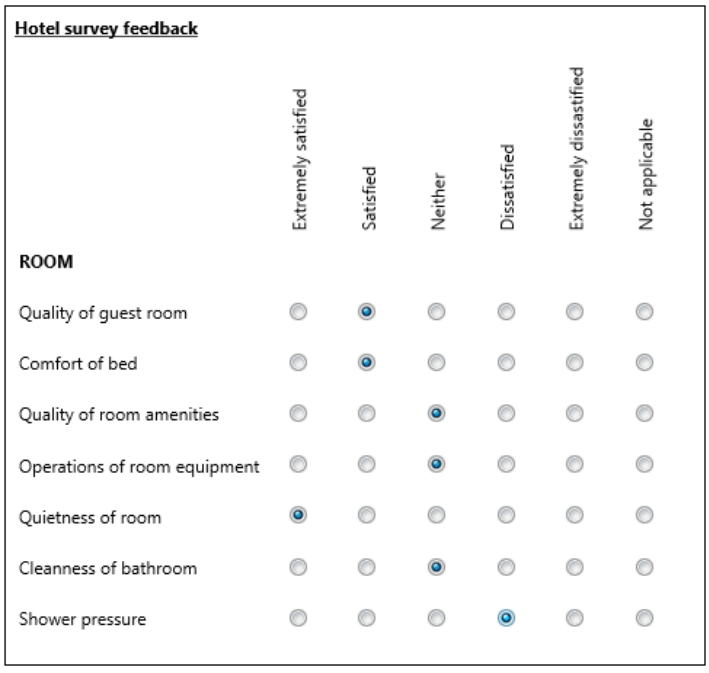
TTMSFMXRatingGrid

The TTMSFMXRatingGrid is a control for capturing ratings for different items or for presenting feature comparison lists.

Features:

- Customizable tickmarks
- Separator items
- Items can be radiogroup for single rate selection or checkgroup for multiple feature selection
- Items & rating categories can be displayed with different font colors & font styles
- Up to 64 rating categories





Properties:

BitmapContainer: TTMSFMXBitmapContainer; Container used for the bitmaps displayed in the control.
Categories: TTMSFMXRatingCategories; List of catogories. Determines the number of displayed columns.
CategoryFont: TFont; The font used for the category titles.
CategoryFontColor: TAlphaColor; The font color used for the category titles.

CategoryType: TTMSFMXCategoryType; Set to catRating if the items should behave as a radiogroup for single rate selection. Set to catFeature to make the items behave as a checkgroup for multiple feature selection.
CategoryOrientation: TTMSFMXCategoryOrientation; Set the orientation of the category titles to vertical or horizontal.
CheckOffBitmap: TBitmap; The bitmap used for an unselected checkbox.
CheckOffBitmapName: String; The bitmapname for the bitmap from the BitmapContainer used for an unselected checkbox.
CheckOnBitmap: TBitmap; The bitmap used for a selected checkbox.
CheckOnBitmapName: String; The bitmapname for the bitmap from the BitmapContainer used for a selected checkbox.
RadioOffBitmap: TBitmap; The bitmap used for an unselected radiobutton.
RadioOffBitmapName: String; The bitmapname for the bitmap from the BitmapContainer used for an unselected radiobutton.
RadioOnBitmap: TBitmap; The bitmap used for a selected radiobutton.
RadioOnBitmapName: String; The bitmapname for the bitmap from the BitmapContainer used for a selected radiobutton.
ItemFont: TFont; The font used for the item titles.
ItemFontColor: TAlphaColor: The font color used for the item titles.
Items: TTMSFMXRatingItems; List of items. Determines the number of displayed rows.
Spacing: single; The space between rows and columns.
DisableFocusEffect: Boolean; Determines if a focusborder is displayed around the focused tickmark.

Events:

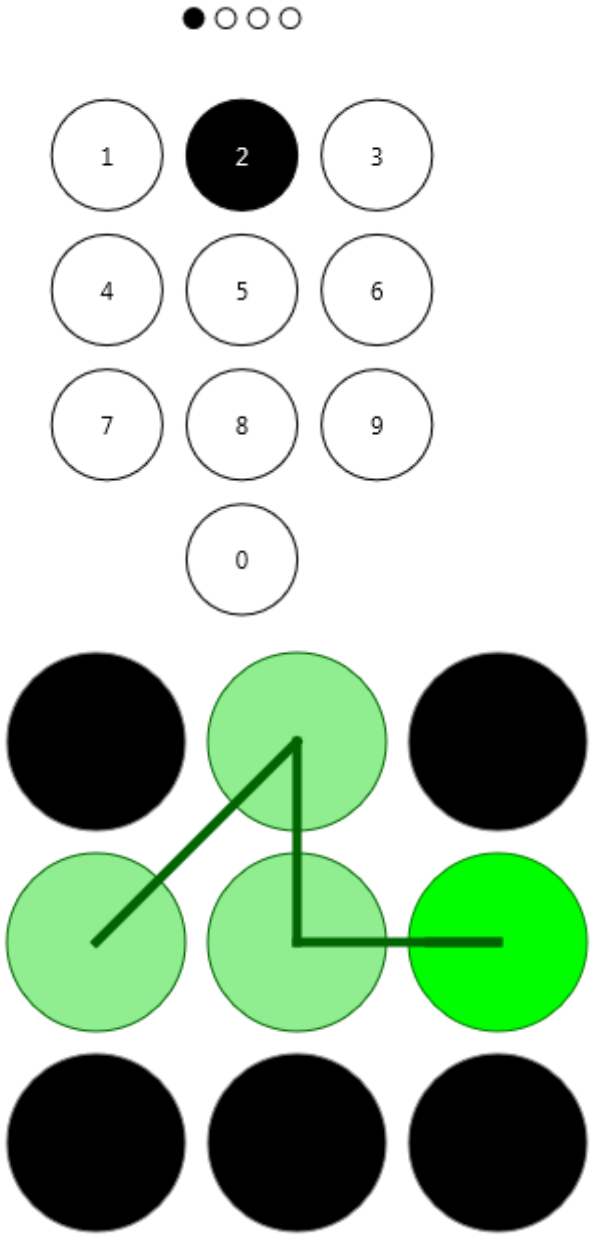
OnValueChanged(Sender: TObject; ItemIndex, CategoryIndex: Integer; Value: boolean);
Event triggered when a tickmark is selected or unselected.

TTMSFMXPassLock

The TTMSFMXPassLock is a control for capturing passwords on a keypad or circle pattern.

Features:

- Number & pattern passlock control
- Pass lock entry & pass lock learn mode
- Customizable appearance
- Configurable pass code length



Properties:

Font: TFont; The font used for the keypad circles.
FontColor: TAlphaColor; The default font color.
HotFontColor: TAlphaColor; The font color for a pressed circle.
HotItemFillColor: TAlphaColor; The fill color for a pressed circle.
HotItemStrokeColor: TAlphaColor; The stroke color for a pressed circle.
ItemFillColor: TAlphaColor; The default circle fill color.
ItemStrokeColor: TAlphaColor; The default circle stroke color.
SelectItemFillColor; TAlphaColor; The fill color for a selected circle.
SelectItemStrokeColor; TAlphaColor; The stroke color for a selected circle.

LearnMode: Boolean; When set to true, enables the user to configure a new password. The OnPasswordLearned event is triggered after the first time a password has been entered. The OnPasswordConfirmed event is triggered after the second time a password is entered.
When set to false, the control only accepts the password as defined in PassValue. The OnPasswordMatch or OnPasswordMismatch event is triggered after a correct or incorrect password has been entered respectively.
LineColor: TAlphaColor; The color of the connecting line (pltPattern mode only).
LineThickness: TAlphaColor; The thickness of the connecting line (pltPattern mode only).
LockType: TAlphaColor; TTMSFMXPassLockType; The type of lock that is displayed. Set to pltNumber to display a keypad or to pltPattern to display circle pattern.
PassLength: integer; Set the length of the password.
PassValue: string; Set the password value. The control accepts a password when it is equal to PassValue.
PassEntry: string; The password that has been entered via the control.
ShowEntry: Boolean; When set to true, displays how many numbers have already been entered on top of the keypad (pltNumber mode only).
Spacing: Single; The space between circles.

Events:

OnValueChanged(Sender: TObject; ItemIndex: Integer);
Event triggered when a circle is pressed.

OnPasswordMatch(Sender: TObject);
Event triggerend when a password is entered that matches the PassValue value (Only when LearnMode is set to false).

OnPasswordMisMatch(Sender: TObject);
Event triggered when a password is entered that does not match the PassValue value (Only when LearnMode is set to false).

OnPasswordLearned(Sender: TObject);
Event triggered when a password is entered for the first time (Only when LearnMode is set to true).

OnPasswordConfirmed(Sender: TObject; Result: TTMSFMXPasswordConfirm);
Event triggered when a password is entered for the first time (Only when LearndMode is set to true). When the second password entry matches the first password entry the Result is pcSuccess and the password value is set to the PassValue, if not the result is pcFail and the password is discarded.

TTMSFMXTouchKeyboard / TTMSFMXPopupTouchKeyboard



Description

TTMSFMXTouchKeyboard is a compact light-weight virtual keyboard for desktop applications on Mac OS-X and Windows, which includes typical keyboards as QWERTY, QWERTZ, AZERTY, DVORAK, Cellphone and Numeric. The possibility exists to use the keyboard as a popup.
The keyboard can be fully customized in order and style.

Properties & Events

TTMSFMXTouchKeyboard

AutoCapsDisplay	If turned on, the keyboard will show key text in uppercase when caps is on
BorderRounding	Sets the roundings of the keys
HighlightCaps	Sets the highlight color for the available capital keys
HighlightAltGr	Sets the highlight color for the available alt keys
BitmapContainer	A bitmap container for image storage
KeyboardType	Selects one of the predefined keyboard types
Keys	Collection of keys in the keyboard
OnKeyClick	Event triggered when a key is clicked
OnKeyDown	Event triggered when mouse is down on the key
OnDrawKey	Event triggered to perform custom key drawing

TTMSFMXPopupTouchKeyboard

AutoCapsDisplay	If turned on, the keyboard will show key text in uppercase when caps is on.
HighlightCaps	Sets the highlight color for the available capital keys
HighlightAltGr	Sets the highlight color for the available alt keys
KeyboardType	Selects one of the predefined keyboard types
OnClose	Event triggered when the popup keyboard closes
OnKeyboardCreated	Event triggered after the internal keyboard is created and before it is shown

TTMSFMXTouchKeyItem

X	The actual x position of the key on the keyboard
Y	The actual y position of the key on the keyboard
Caption	The normal key state caption

ShiftCaption	The key caption while pressing shift
AltGrCaption	The key caption while pressing alt
KeyValue	The key value in normal state
ShiftKeyValue	The key value while pressing shift
AltGrKeyValue	The key value while pressing alt
PictureDownState	The picture shown when the key is being pressed
PictureNormalState	The picture in normal state
SpecialKey	If this key is special, for example the ctrl key
BorderColor	The border color in normal state
borderColorDown	The border color when the key is being pressed
Color	The color in normal state
ColorDown	The color when the key is being pressed
TextColor	The text color
TextColorDown	The text color when the key is being pressed
ImageName	The name of the image to use from the assigned bitmapcontainer
ShortCut	The keys' shortcut
Hint	The hint that's shown when hovering over a key

Methods

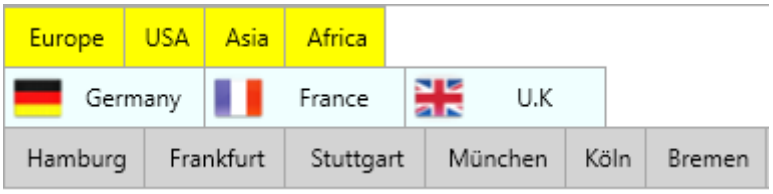
TTMSFMXTouchKeyboard

SaveKeybdLayout	Saves the keyboard to the specified file name
LoadKeybdLayout	Loads the keyboard from the specified file name

TTMSFMXPopupTouchKeyboard

ShowKeyboard	Shows the keyboard on the cursors position

TTMSFMXScrollMenu



Description

TTMSFMXScrollMenu is a compact horizontal multi-level menu scroller with a vertical hierarchic model for multiple layers of the menu.

Properties & Events

TTMSFMXScrollMenu

Items	A collection of scroll items
BitmapContainer	Reference to bitmap container that can contain images to be used for the menu items
OnItemAppearance	Event triggered when the item appears on screen and allows appearance customization
OnSelect	Event triggered when an item is selected

TTMSFMXScrollItems

AutoSize	Makes the item adapt its size to the text it contains
Appearance	Class property holding all attributes for customizing the appearance of the items
Tag	Integer reference property

TTMSFMXItemAppearance

DefaultWidth	The size of the item, if autosize is false.
Selected/Disabled/-Fill	Fill appearance setting for selected and disabled state
Selected/Disabled/-Stroke	Stroke appearance settings
Selected/Disabled/-Font	Font for selected and disabled state
HorizontalTextAlign	The horizontal text alignment
VerticalTextAlign	The vertical text alignment
Disabled/Selected/-TextColor	The color of the text for selected and disabled state
OnChange	Event triggered when the appearance settings have changed

TTMSFMXScrollItem

BitmapName	(When the BitmapContainer is assigned) The name of the corresponding bitmap in the BitmapContainer to use for the item
SubItems	A collection of the sub items for the item

State	The current state of the item (selected, normal, disabled)
Tag	The numeric representation of the item
Text	The caption of the item

Methods

TTMSFMXScrollMenu

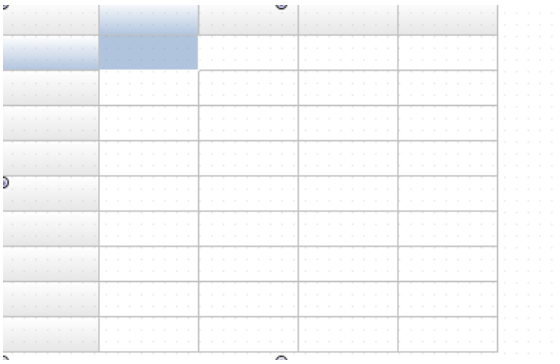
AddItem(AText: string): TTMSFMXScrollItem	Adds an item with the specified text

TTMSFMXScrollItem

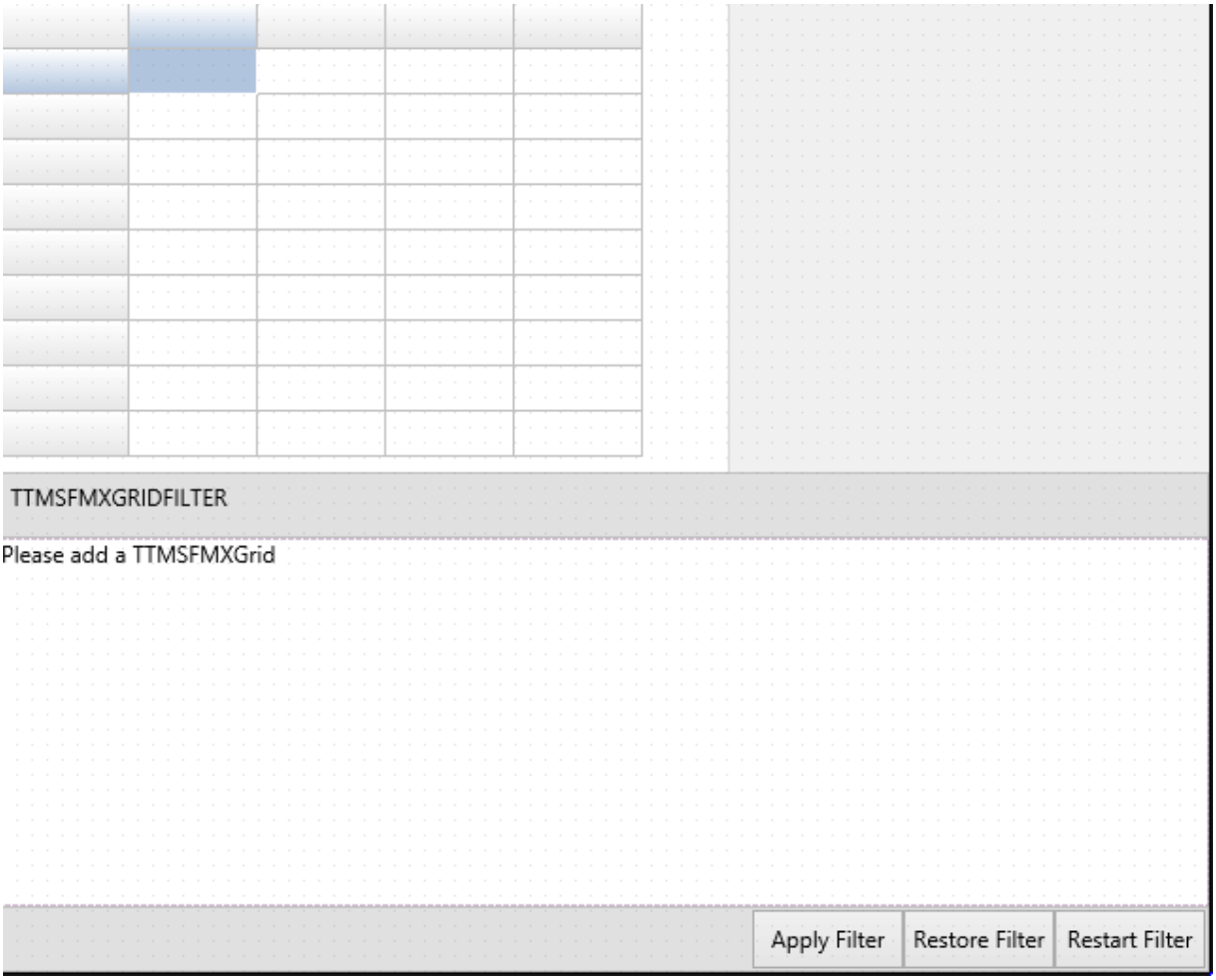
AddSubItem(AText: string): TTMSFMXScrollItem	Adds a subitem with the specified text

TTMSFMXGridFilterPanel

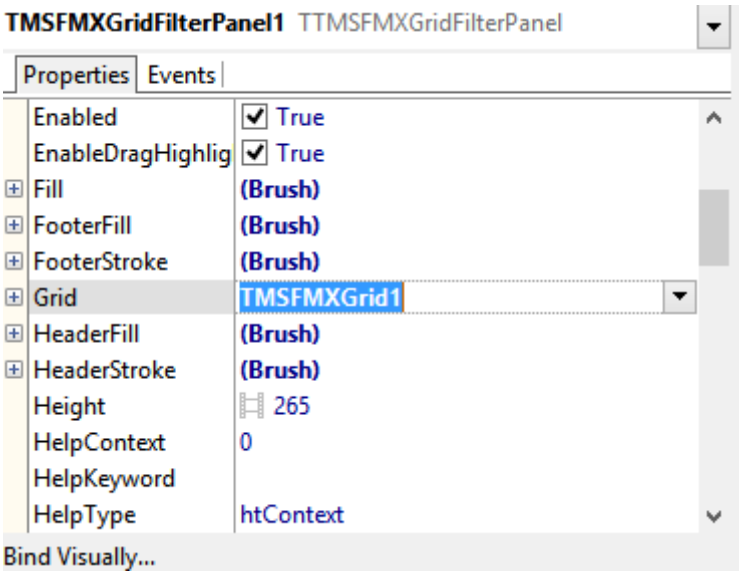
The TTMSFMXGridFilter applies a filter to TTMSFMXGrid, so first drop a TTMSFMXGrid on the form.



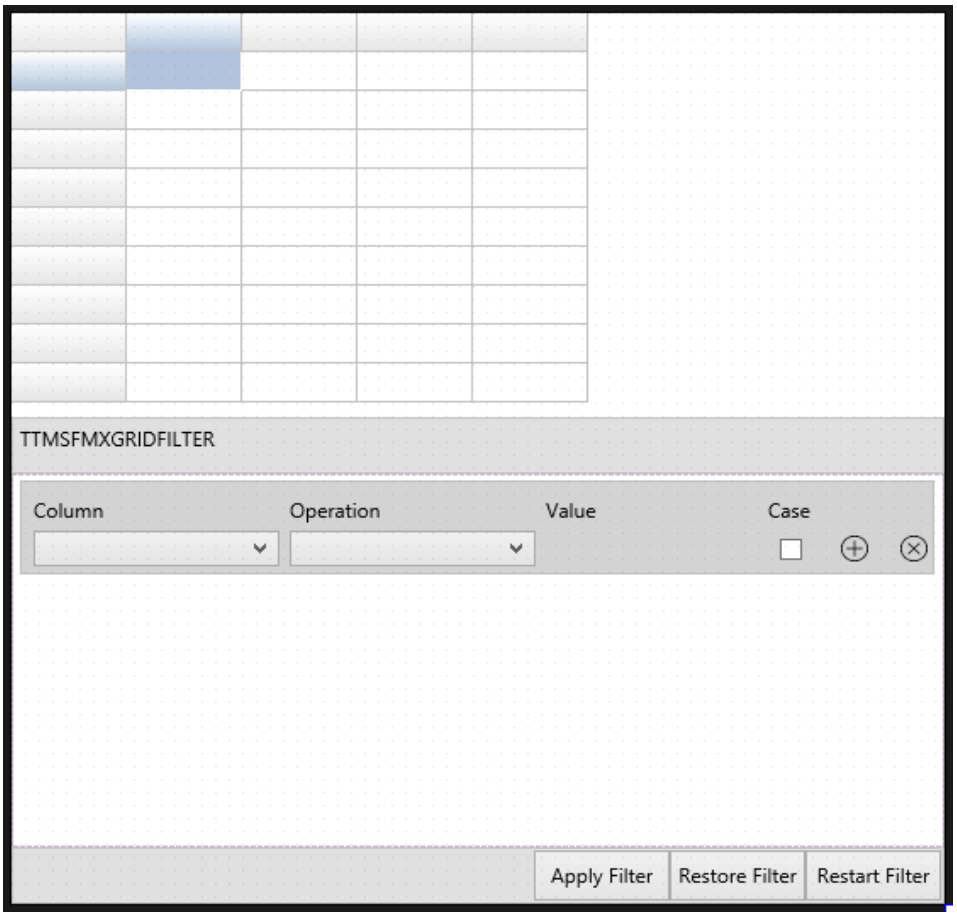
Next drop the TTMSFMXGridFilterPanel on the form.



The component gives the hint to assign a TTMSFMXGrid to the panel:

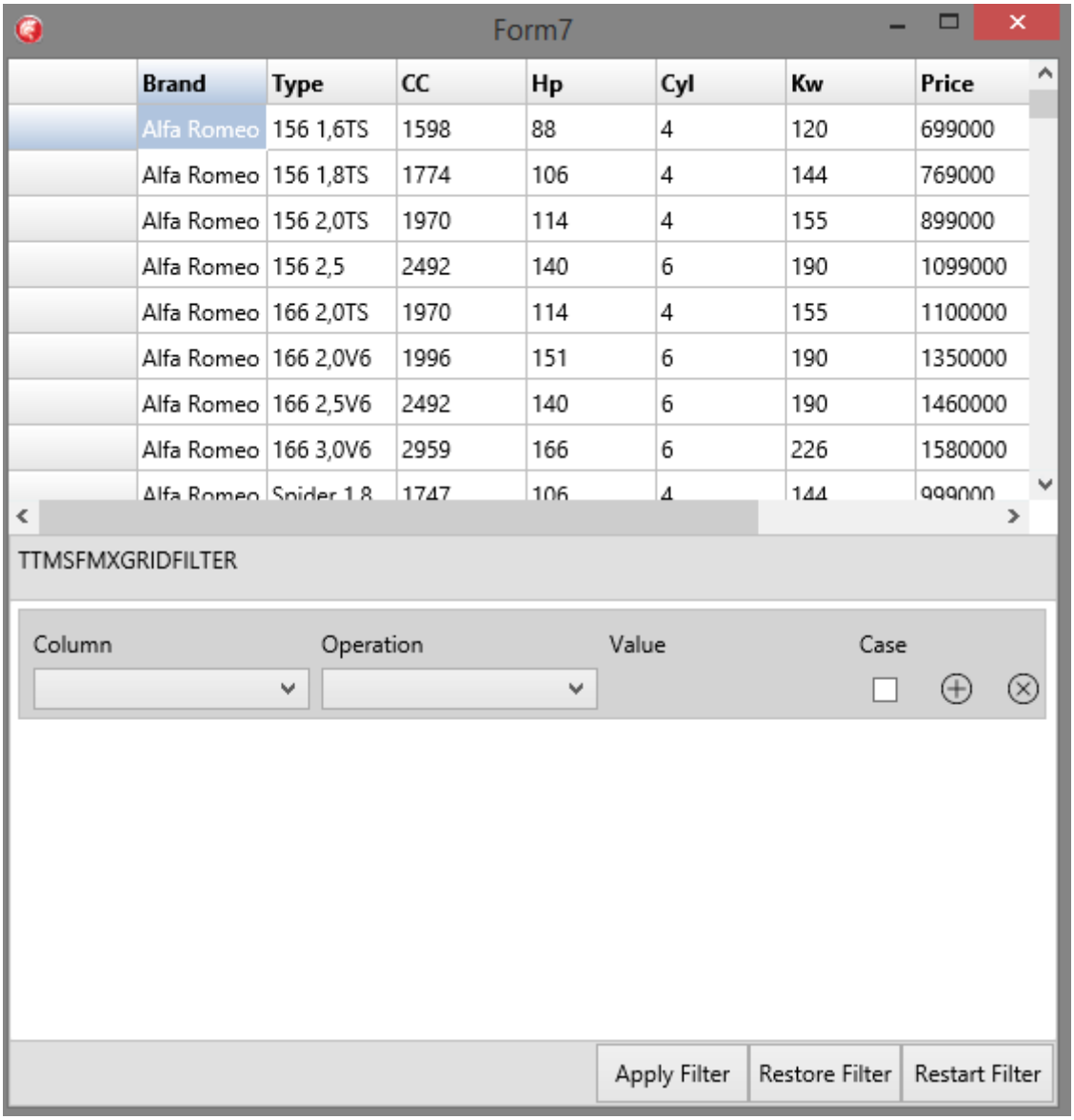


The component will automatically retrieve column information after assignment of the grid:



If preferred, there's a possibility to change the full color and text layout properties of the component. See the full properties list for more explanation.

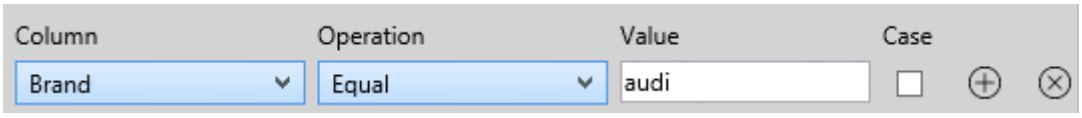
Runtime



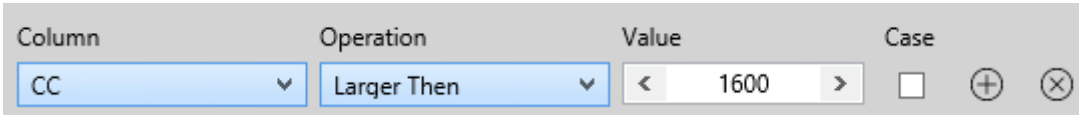
The filter starts with a clean row. All assigned grid columns are available in the column TComboBox. When a column has been chosen, the filter operation will automatically show the first option for a quick and simple experience.
The value and case field will change, depending on the selected column field.

Example:

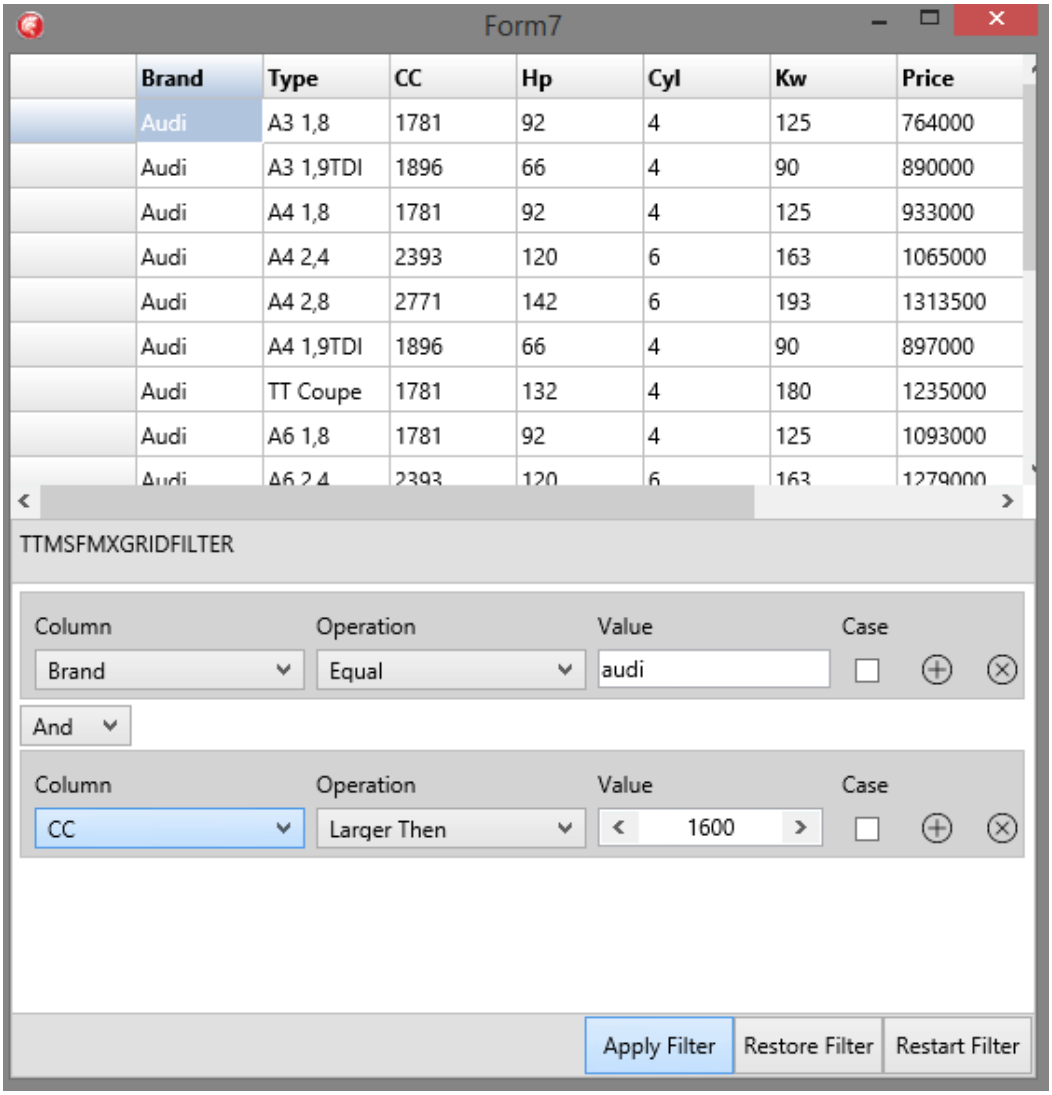
- The column field contains text: The value field became a TEdit



- The column field contains a numeric value: The value field became a TSpinBox



The result so far:



The grid filter was changed after clicking the "*Apply Filter*" button. The same can be accieved for mobile devices.

Properties

ApplyButton	Gives access to the apply button
ButtonAddIcon	Gives access to the add item icon
ButtonRemoveIcon	Gives access to the remove item icon
ClearButton	Gives access to the clear button
Fill	Can fill the component
Footer	Gives access to the components footer
FooterAddClearButton	Gives access to the footers clear button
FooterAddFilterButton	Gives access to the footers add filter button
FooterRestoreFilterButton	Gives acces to the footers restore filter button
FooterFill	Can fill the components footer
FooterStroke	Can give a stroke to the footers border
Grid	Gives access to the assigned grid
Header	Gives access to the components header
HeaderTitle	Gives access to the header title
HeaderFill	Can fill the header
HeaderStroke	Can give a stroke to the headers border
ItemFill	Can fill the items
ItemStroke	Can give a stroke to the items border
LabelFont	Gives access to all label fonts
LabelFontColor	Gives access to all labels font colors
RestoreButton	Gives access to the restore button
Stroke	Can give a stroke to the components border
ShowRestoreFilter	Gives access to the visibility of the restore filter button
ShowApplyFilter	Gives access to the visibility of the apply filter button
TitleFont	Gives access to the title font
TitleFontColor	Gives access to the title font color
Title	Gives access to the title caption
UI	Gives access to all text properties
UIType	Gives access to the types of UI

ApplyButton	Sets the text for the apply button
ApplyDialogText	Sets the confirmation dialog text for the apply button
ColumnLabel	Sets the text for the column label
ColumnHint	Sets the column label hint text
CaseLabel	Sets the text for the case label
CaseHint	Sets the case label hint text
ClearButton	Sets the text of the “clear filter” button
ClearDialogText	Sets the text of the “clear filter” dialog
HintApplyFilter	Sets the apply filter hint text
HintClearFilter	Sets the clear filter hint text
HintFilterAdd	Sets the add filter button hint text
HintFilterRemove	Sets the remove filter button hint text
HintRestoreFilter	Sets the restore filter button hint text
OperationLabel	Sets the text of the operation label
OperationHint	Sets the text of the operation hint
OperationEqual	Sets the text of the “equal” operation item
OperationNotEqual	Sets the text of the “not equal” operation item
OperationContains	Sets the text of the “contains” operation item
OperationBeginsWith	Sets the text of the “begins with” operation item
OperationEndsWith	Sets the text of the “ends with” operation item
OperationSmallerThen	Sets the text of the “smaller then” operation item
OperationLargerThen	Sets the text of the “larger then” operation item
OperationSmallerOrEqual	Sets the text of the “smaller or equal” operation item
OperationLargerOrEqual	Sets the text of the “larger or equal” operation item
OperationTrueFalse	Sets the text of the “true false” operation item
OperationAnd	Sets the text of the “and” action item
OperationOr	Sets the text of the “or” action item
RestoreButton	Sets the text of the “restore filter” button
RestoreDialogText	Sets the text of the “restore filter” dialog
ValueLabel	Sets the text of the value label
ValueTextHint	Sets the text of the value label hint

Events

UI.OnChange	Triggers when changing a ui item
OnAppliedFilter	Triggers when the filter was applied

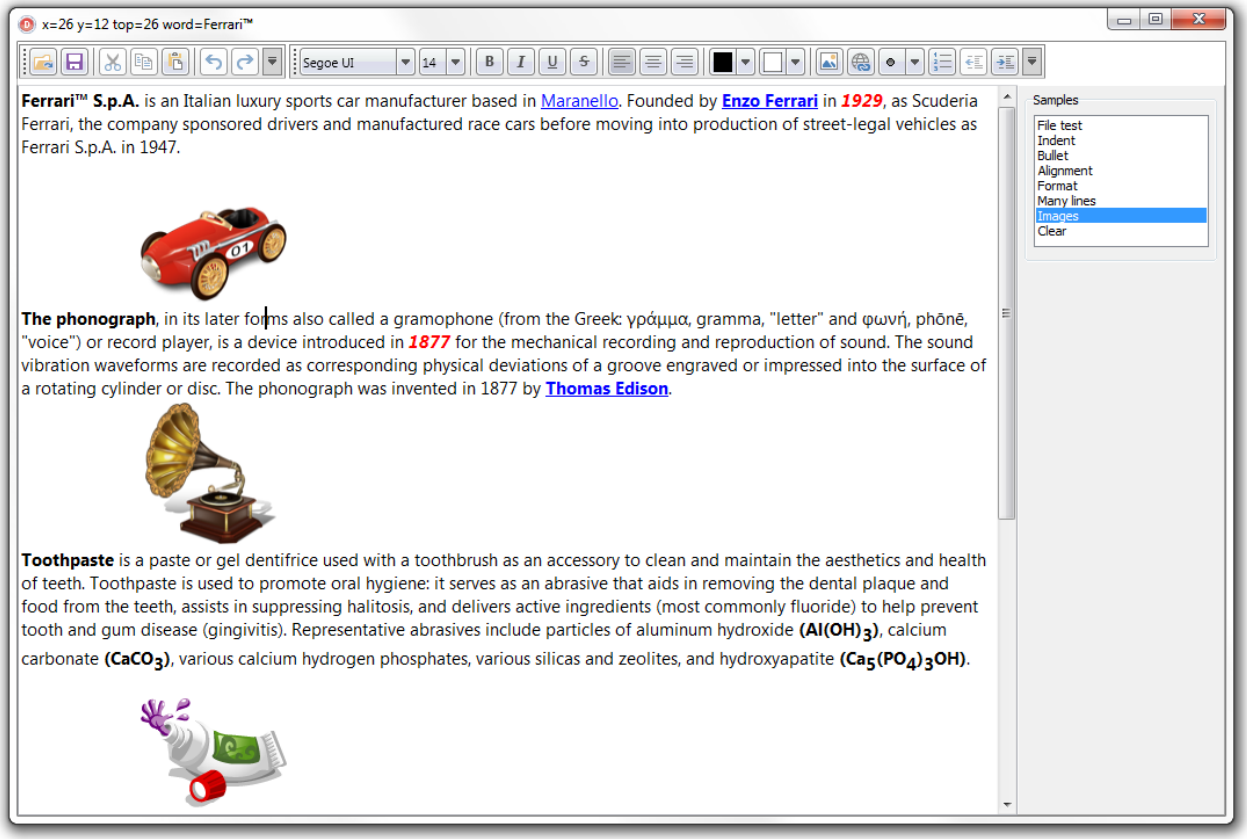
Methods

Init	Initiates the grid filter

TTMSFMXRichEditor

Description

TTMSFMXRichEditor is a compact light-weight WYSIWYG editor for formatted text. TTMSFMXRichEditor can include formatted text with bullets, hyperlinks, images, indenting, and aligned paragraphs. It offers functions for merging, highlighting text, find & replace, undo/redo, clipboard.
TTMSFMXRichEditor stores its text natively in the .RTE file format. It can load text from .TXT and .RTE files and can export to .TXT, .RTF, .HTML and .RTE files. Rich editing/formatting toolbars are included to perform clipboard functions, undo/redo, formatting, paragraph alignment, inserting bullets, pictures, hyperlinks, special characters.



Organization

The core component is TTMSFMXRichEditor. This is a standalone component that can be used as-is for WYSIWYG editing of formatted text. It comes with a formatting and editing toolbar that can be used to quickly setup a rich editor or its many predefined toolbar buttons/pickers can be used to create a specific user interface around the TTMSFMXRichEditor according to your needs.

Internally the TTMSFMXRichEditor consists of a simple DOM. This DOM is a generic list of document elements. Different types of document elements exist such as a text element, image element, linebreak element, bullet element, … Each document element has several attributes that determine the appearance in the document. While the TTMSFMXRichEditor provides a large series of methods to add or remove elements from the DOM, it is also accessible via

TTMSFMXRichEditor.Context.Content. It is recommended though that the API used instead of direct DOM manipulation.

Getting Started

Drop a TTMSFMXRichEditor on the form. The component with its default settings is ready for use. Entering of text can be done with default font & alignment. For ease of use, connect a TTMSFMXRichEditorEditToolBar or TTMSFMXRichEditorFormatToolBar, to apply all kinds of formatting to the text without writing any code or use its ribbon equivalents for a WYSIWYG editor with toolbar UI.

Properties & Events

Properties

Author	Sets the author of the document that will be persisted when saving to .RTE file format.
Color	Sets the default background of the TTMSFMXRichEditor
Comments	Sets comments for the document that will be persisted when saving to .RTE file format.
FontColor	Sets the default font color of the TTMSFMXRichEditor
GraphicSelection	Sets the appearance of the grips that appear when selecting graphics in the TTMSFMXRichEditor
GraphicSelection.BorderColor	Sets the border color of graphic item grips
GraphicSelection.Color	Sets the background color of graphic item grips
GraphicSelection.Style	Selects the style between rectangular or circular for the grips
HighlightColor	Sets the background color for highlighted text in the TTMSFMXRichEditor
HighlightTextColor	Sets the text color for highlighted text in the TTMSFMXRichEditor
LastModifiedBy	Sets the name of the person who last modified the content of the document and this name Is persisted in the .RTE file
ReadOnly	When true, the content of the document cannot be altered but selection is possible
SelectionColor	Sets the background color for selection in the TTMSFMXRichEditor
SelectionTextColor	Sets the text color for selection in the TTMSFMXRichEditor
Tags	Sets tags for the document that will be persisted when saving to .RTE file format.
URLColor	Sets the text color for hyperlinks in the TTMSFMXRichEditor
Version	Read-only property returning the version of the component

Events

OnCaretChanged	Event triggered whenever the caret changes in the TTMSFMXRichEditor
OnClick	Event triggered when the editor is clicked
OnClickHyperlink	Event triggered when a hyperlink is clicked in the

	editor. The URL for the hyperlink is returned as a parameter
OnDrawGraphic	Event triggered for drawing custom graphic elements in the TTMSFMXRichEditor. This event returns the canvas and rectangle where to draw the custom graphic and an ID for the graphic element
OnEnter	Event triggered when the TTMSFMXRIchEditor gets focus
OnEnterWord	Event triggered when one or more characters were entered before a word boundary. The event returns the word just entered
OnExit	Event triggered when the TTMSFMXRIchEditor looses focus
OnSelectionChanged	Event triggered whenever the selection changes in the TTMSFMXRichEditor

Methods

AddBullet(AType: TBulletType = btCircle);	Appends a bullet element to the TTMSFMXRichEditor and returns a bullet document element. The bullet types can be: - btSquare - btCircle - btArrow - btStar - btTick
AddGraphic(AWidth, AHeight: integer; AID: string);	Appends a graphical element with a specific ID to the TTMSFMXRichEditor and returns a graphic document element. This graphical element needs to be drawn via the OnDrawGraphic event
AddHyperlink(AValue, AURL: string);	Sets a hyperlink for the currently selected text in the TTMSFMXRichEditor
AddImage(FileName: string); overload;	Appends an image from file to the TTMSFMXRichEditor and returns a graphic document element
AddImage(FileName: string; AWidth, AHeight: integer); overload;	Appends an image from file with a specific width and height to the TTMSFMXRichEditor and returns a graphic document element
AddImage(Picture: TPicture); overload;	Appends an image to the TTMSFMXRichEditor and returns a graphic document element. Images of the type BMP,JPEG,GIF,PNG,ICO are supported.
AddImage(Picture: TPicture; AWidth, AHeight: integer); overload;	Appends an image with a specific width and height to the TTMSFMXRichEditor and returns a graphic document element. Images of the type BMP,JPEG,GIF,PNG,ICO are supported.
AddLineBreak: TREElement;	Appends a linebreak to the TTMSFMXRichEditor and returns a linebreak document element
AddMultiLineText(AValue: string);	Appends multiple lines of text as word-wrapped text in the TTMSFMXRichEditor
AddText(AValue: string): TTextElement; overload;	Appends text to the TTMSFMXRichEditor and returns a text document element containing this added text
AddText(AValue: string; AAlignment: TAlignment): TTextElement; overload;	Appends text with a specific alignment to the TTMSFMXRichEditor and returns a text document

	element containing this added text
AddText(AValue: string; AColor: TColor): TTextElement; overload;	Appends text with a specific text color to the TTMSFMXRichEditor and returns a text document element containing this added text
AddText(AValue: string; AColor: TColor; BkColor: TColor): TTextElement; overload;	Appends text with a specific text color and background color to the TTMSFMXRichEditor and returns a text document element containing this added text
AddText(AValue: string; AFont: TFont): TTextElement; overload;	Appends text with a specific font setting to the TTMSFMXRichEditor and returns a text document element containing this added text
AddText(AValue: string; AFontSize: integer; AFontName: string; AFontStyle: TFontStyles): TTextElement; overload;	Appends text with a specific font setting to the TTMSFMXRichEditor and returns a text document element containing this added text
AddText(AValue: string; AFontSize: integer; AFontName: string; AFontStyle: TFontStyles; AAlignment: TAlignment): TTextElement; overload;	Appends text with a specific font setting and alignment to the TTMSFMXRichEditor and returns a text document element containing this added text
BeginUpdate;	Use to block updates when doing many programmatic manipulations in the TTMSFMXRichEditor
CanRedo: boolean;	Returns true when a Redo operation is possible
CanUndo: boolean;	Returns true when an Undo operation is possible
CanUnindent: boolean;	Returns true when the selection in the document is indented (and thsu can be unindented)
Clear;	Removes all elements from the document
ClearSelection;	Clears the selection in the document
DeleteCaretElement;	Deletes the document element where the caret is
DeleteChar;	Deletes the character at caret position
DeleteSelected;	Deletes the selected element in case an image or graphical element is selected
DeleteSelection;	Deletes the selection in the TTMSFMXRichEditor
EndUpdate;	Use to block updates when doing many programmatic manipulations in the TTMSFMXRichEditor
FindFirst(AText: string; MatchCase: boolean = false): boolean;	Finds the first occurrence of text from the document origin
FindNext: boolean;	Finds the next occurrence of text from the position of the last find operation
GetSelectionBkColor: TColor;	Returns the background color for the selected text
GetSelectionBullet: TBulletType;	Returns the bullet type used for the selected text
GetSelectionFontName: string;	Returns the font face name for the selected text
GetSelectionFontSize: integer;	Returns the font size for the selected text
GetSelectionIndent: integer;	Returns the indent of the selected text
GetSelectionTextColor: TColor;	Returns the text color for the selected text
GetWordAndIndexAtCaret(var AValue: string; var AIndex: integer);	Returns the word at caret position and the index of the element containing the word
HasSelection: boolean;	Function returns true when there is a selection in the TTMSFMXRichEditor
Highlight(AText: string; MatchCase: boolean = false): boolean;	Highlight the text in the document with or without case sensitivity in the document
InsertBullet(AType: TBulletType = btCircle);	Inserts a bullet element at caret position in the TTMSFMXRichEditor and returns a bullet document element

InsertChar(ch: char);	Inserts a character at caret position
InsertFromStream(const AStream: TStream; f: double);	Inserts plain text from file at caret position
InsertGraphic(ID: string; AWidth, AHeight: integer);	Inserts a custom graphic element with a specific width and height at caret position in the TTMSFMXRichEditor and returns a graphic document element
InsertImage(FileName: string; AWidth: integer = 0; AHeight: integer = 0); overload;	Inserts an image with a specific width and height at caret position in the TTMSFMXRichEditor and returns an image document element
InsertImage(Picture: TPicture; AWidth: integer = 0; AHeight: integer = 0); overload;	Inserts an image with a specific width and height at caret position in the TTMSFMXRichEditor and returns an image document element
InsertMultiLineText(AValue: string);	Inserts text in the TTMSFMXRichEditor at caret position
InsertText(AValue: string): TTextElement; overload;	Inserts text in the TTMSFMXRichEditor at caret position and returns a text document element containing this added text
InsertText(Index: integer; AValue: string): TTextElement; overload;	Inserts text in the TTMSFMXRichEditor at document element Index and returns a text document element containing this added text
IsCaretInBulletList(var AType: TBulletType; var AIndex, AIndent: integer): boolean;	Returns true when the caret is within a list of bulleted items and when so, returns the bullet type, the index of the item in the list and the indent of the bulleted items
IsEmpty: boolean;	Returns true when the document is empty
IsSelectionBold: boolean;	Returns true when the selected text font style is bold
IsSelectionCenter: boolean;	Returns true when the selected text alignment is center aligned
IsSelectionItalic: boolean;	Returns true when the selected text font style is italic
IsSelectionLeft: boolean;	Returns true when the selected text alignment is left aligned
IsSelectionRight: boolean;	Returns true when the selected text alignment is right aligned
IsSelectionStrikeOut: boolean;	Returns true when the selected text font style is strikeout
IsSelectionSubscript: boolean;	Returns true when the selected text font style is subscript
IsSelectionSuperscript: boolean;	Returns true when the selected text font style is superscript
IsSelectionUnderline: boolean;	Returns true when the selected text font style is underline
LoadFromFile(const FileName: string);	Load a document from the .RTE file format
LoadFromStream(const AStream: TStream);	Load a document in the .RTE file format from stream
LoadFromTextFile(const FileName: string);	Loads the document from a plain text file
Merge(NamesAndValues: TStringList);	Performs merging of mergefields with merge values contained in the stringlist
PlainText: string;	Returns the text of the TTMSFMXRichEditor document as plaintext
property Caret: TCaret read FCaret write FCaret;	Allows to get and set the caret based on document elements and character position within the selected document element

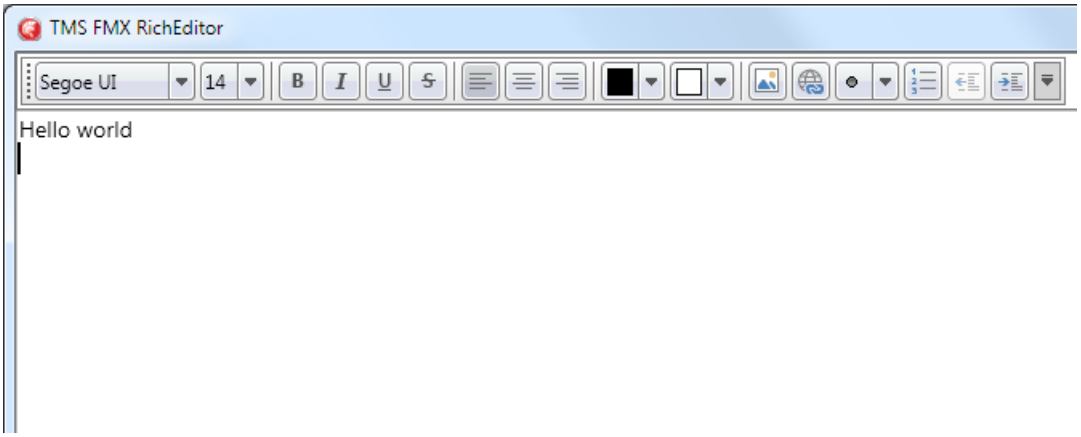
property Selected: TREElement read FSelected write FSelected;	Get or set the selected (graphical) document element
property Selection: TSelection read FSelection write FSelection;	Allows to get and set the selection in the TTMSFMXRichEditor based on document elements for the selection start and selection end and character positions within the selections
Redo;	Performs Redo
ReplaceFirst(AText, AReplacement: string; MatchCase: boolean = false): boolean;	Replaces the first occurrence of text from the document origin
ReplaceNext: boolean;	Replaces the next occurrence of text from the position of the last find operation
SaveSelectionToStream(const AStream: TStream);	Saves the current selected document elements in .RTE file format to stream
SaveToFile(const FileName: string);	Save a document to the .RTE file format
SaveToStream(const AStream: TStream);	Save a document in the .RTE file format to stream
SaveToText(AFileName: string);	Saves the document in TTMSFMXRichEditor as plain text
ScrollToCaret;	Vertically scroll the TTMSFMXRIchEditor to make the caret visible
SelectAll;	Selects all document elements in TTMSFMXRichEditor
SelectedText: string;	Returns the selected text
SelectText(FromChar, ALength: integer);	Selects text in the TTMSFMXRichEditor based on character position of the text and length in characters
SelectWordAtCaret: string;	Selects the word in the TTMSFMXRichEditor document at caret position
SelectWordAtXY(X,Y: integer): string;	Selects the word in the TTMSFMXRichEditor document at mouse coordinates X,Y
SetSelectionAttribute(AAlignment: TAlignment); overload;	Sets the alignment of the selected text
SetSelectionAttribute(AError: boolean); overload;	Sets the selected text with red error underlining or remove error underlining
SetSelectionAttribute(AFont: TFont; AColor: TColor); overload;	Sets the font and color attribute of the seleted text
SetSelectionAttribute(AFont: TFont; AColor: TColor; BkColor: TColor); overload;	Sets the font, text color and background color attribute of the seleted text
SetSelectionAttribute(AFontName: string; AFontSize: integer; AFontStyle: TFontStyles; AColor: TColor); overload;	Sets the font and color attribute of the seleted text
SetSelectionAttribute(AFontName: string; AFontSize: integer; AFontStyle: TFontStyles; AColor, BkColor: TColor); overload;	Sets the font, text color and background color attribute of the seleted text
SetSelectionBkColor(AColor: TColor);	Sets the background color of the selected text
SetSelectionBold(DoBold: boolean);	Sets the selected text bold or remove bold
SetSelectionBullets(AType: TBulletType); overload;	Sets bullets for the selected text. Each line separated by a linebreak gets a bullet. AType sets the bullet type
SetSelectionColor(AColor: TColor);	Sets the text color of the selected text
SetSelectionError(DoError: boolean);	Sets the selected text with red error underlining or remove error underlining
SetSelectionFontName(AName: string);	Sets the font face name for the selected text
SetSelectionFontSize(ASize: integer);	Sets the font size for the selected text
SetSelectionHighlight;	Sets the selected text in highlight text /

	background colors
SetSelectionHyperlink(AURL: string);	Sets a hyperlink for the text selected element in the document
SetSelectionIndent(AIndent: integer);	Sets the indent on the selected text
SetSelectionItalic(DoItalic: boolean);	Sets the selected text italic or remove italic
SetSelectionMergeField(AMergeName: string);	Defines a mergefield value for the selected text
SetSelectionStrikeOut(DoStrikeOut: boolean);	Sets the selected text strikeout or remove strikeout
SetSelectionSubscript(DoSubScript: boolean);	Sets the selected text subscript or remove subscript
SetSelectionSuperscript(DoSuperScript: boolean);	Sets the selected text superscript or remove superscript
SetSelectionUnderline(DoUnderline: boolean);	Sets the selected text underlined or remove underlined
Undo;	Performs Undo
UnHighlight;	Undo any previous highlight
UnSelect;	Undo any selection in the document
UpdateWordAndIndexAtCaret(AValue: string; AIndex: integer);	Replaces the word at document element at caret position at character index AIndex by AValue
WordAtCaret: string;	Returns the word at caret position
WordAtXY(X,Y: integer): string;	Returns the word at X,Y mouse coordinates
XYToCaret(X,Y: integer); overload;	Sets the caret at mouse X,Y coordinates
XYToCaret(X,Y: single); overload;	
XYToChar(X,Y: integer; el: TREElement; var CX,CY: integer): integer;	Converts the X,Y mouse coordinates to character position in the document text
XYToElement(X,Y: integer; var el: TREElement): boolean;	Retrieves the document element at mouse X,Y coordinates
XYToWord(X,Y: integer): string; overload;	Returns the word at mouse coordinates X,Y
XYToWord(X,Y: integer; el: TREElement): string; overload;	Returns the word and document element at mouse coordinates X,Y

Programmatic access to the document

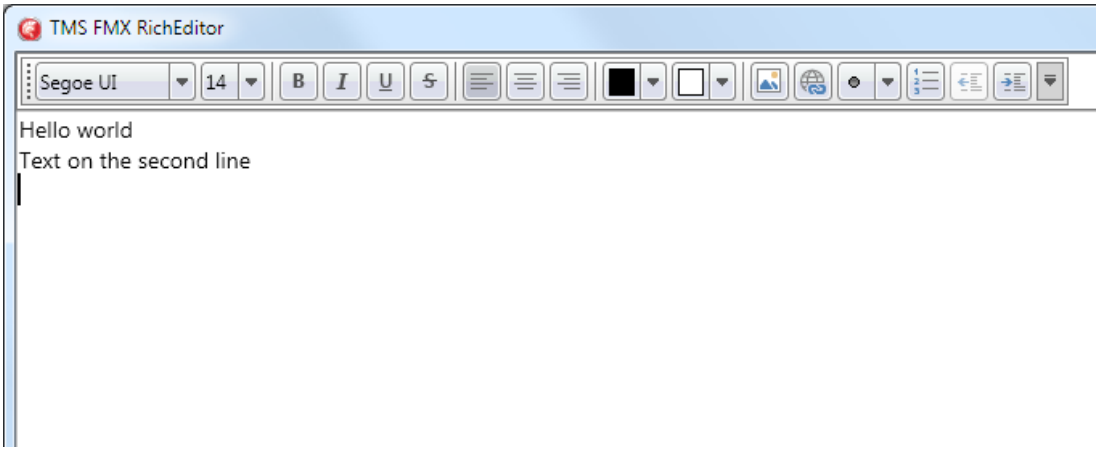
Text can be inserted in TTMSFMXRichEditor in various ways. To start with call:

```
TMSFMXRichEditor1.AddText('Hello world');
```



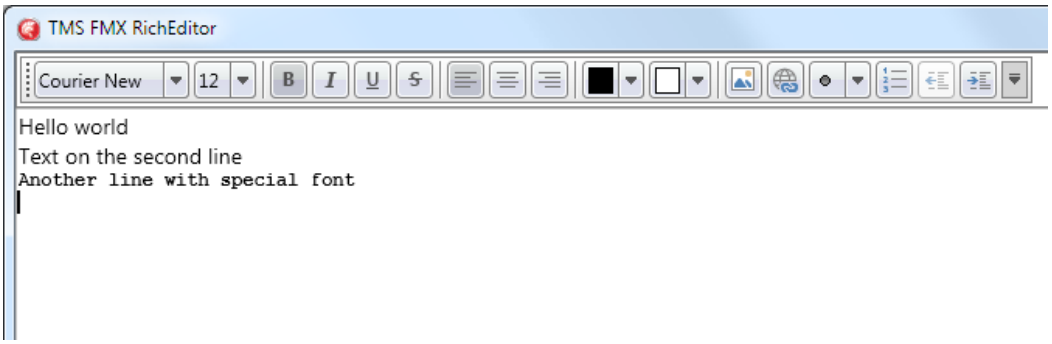
Add text on the next line with:

```
TMSFMXRichEditor1.AddLineBreak;
TMSFMXRichEditor1.AddText('Text on the second line');
```



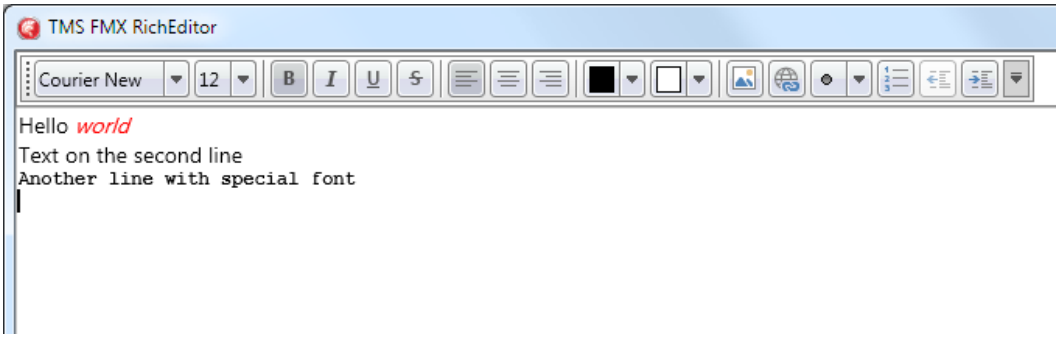
To add text with a different font than default font, use:

```
TMSFMXRichEditor1.AddLineBreak;
TMSFMXRichEditor1.AddText('Another line with special
font',12,'Courier',[fsBold]);
```



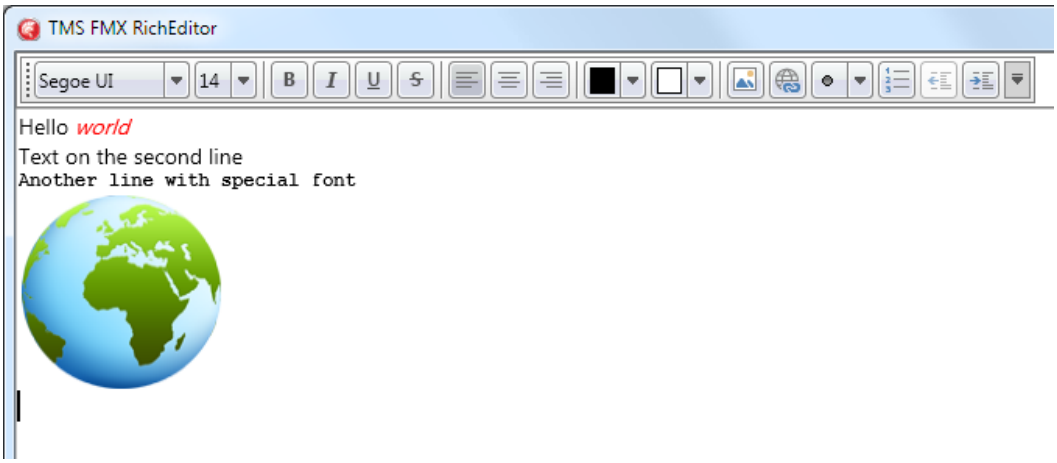
To change attributes of text in the TTMSFMXRichEditor, perform a selection based on index of the text and length. For example, to change the color of “world” on the first line, set a selection from character 6 for 5 characters (character index starts at zero) and set an attribute for the selection followed by remove the selection itself:

```
TMSFMXRichEditor1.SelectText(6,5);
TMSFMXRichEditor1.SetSelectionColor(claRed);
TMSFMXRichEditor1.SetSelectionItalic(True);
TMSFMXRichEditor1.ClearSelection;
```



To add images to the TTMSFMXRichEditor, use:

```
TMSFMXRichEditor1.AddImage('.\sample.png');
```



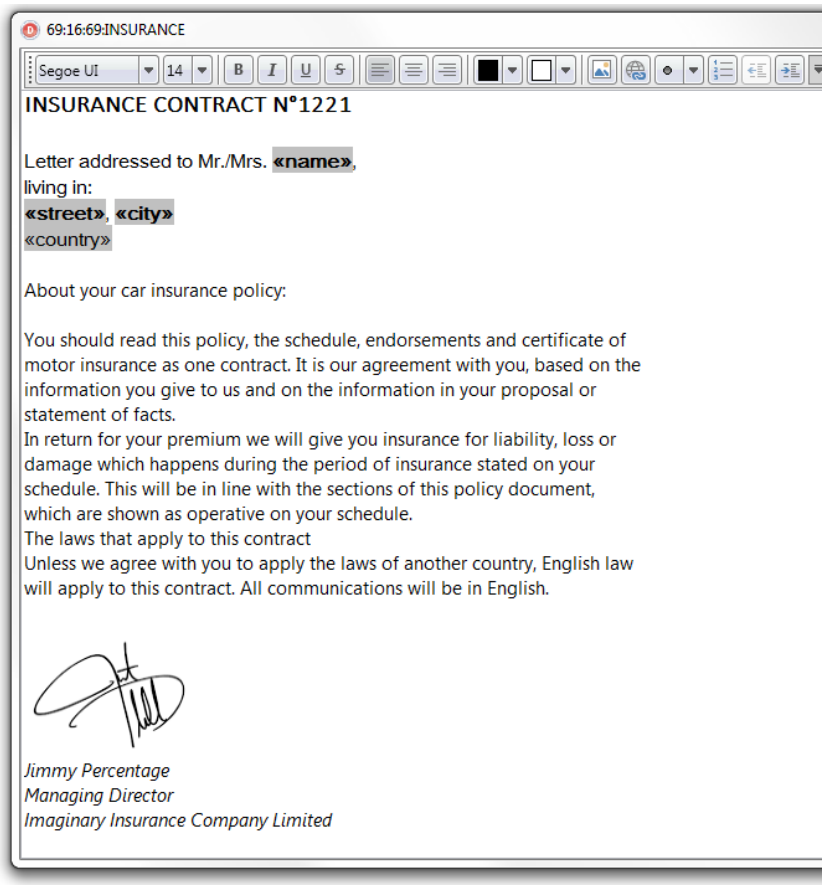
Using merge fields

Via merge fields, specific places in the document can be quickly replaced during a merge operation. To perform merging, first insert merge fields in the document. Merge fields are pieces of text that get a merge field name. These pieces of text are displayed between brackets «» and with a gray background. To set a piece of text as merge field, select the text and call

```
TMSFMXRichEditor1.SetSelectionMergeField('MergeFieldName');
```

Assume that following merge field names exist in the TTMSFMXRichEditor document:

'Name'
'Street'
'City'
'Country'



then a merge operation can be done in the following way:

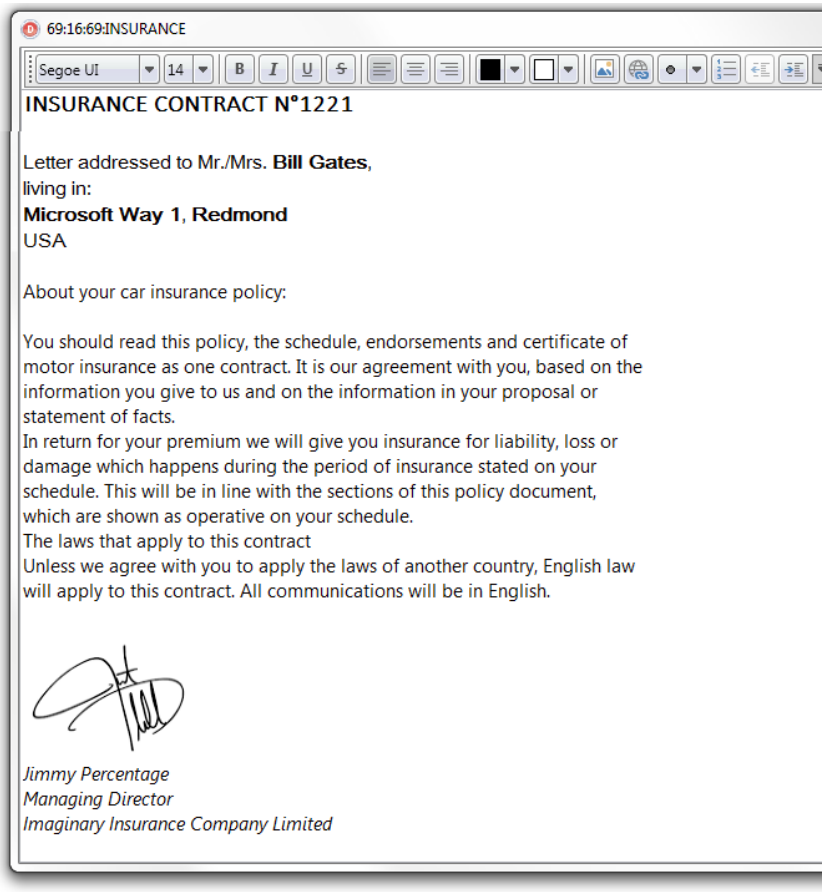
```
var
  sl: TStringList;

sl := TStringList.Create;
sl.Add('Name=Bill Gates');
sl.Add('Street=Microsoft Way 1');
sl.Add('City=Redmond');
sl.Add('Country=USA');
```

```
TMSFMXRichEditor1.Merge(sl);

sl.Free;
```

This will replace the merge fields Name, Street, City, Country with the values 'Bill Gates', 'Microsoft Way 1', 'Redmond', 'USA' specifically.



It is also possible to replace merge fields by pictures, i.e. insert pictures dynamically during a merge operation.

To do this, set a merge fieldname just like for text but using following construct for the mergelist:

Assume that in the previous example we want to add a picture of the person in the document, this would become:

'Photo'
'Name'
'Street'
'City'
'Country'

A merge operation is done the following way:

```
var
  sl: TStringList;
  pic: TBitmap;

pic := TBitmap.Create;
pic.LoadFromFile('billgates.jpg');

sl := TStringList.Create;
sl.AddObject('Photo=',pic);
sl.Add('Name=Bill Gates');
sl.Add('Street=Microsoft Way 1');
sl.Add('City=Redmond');
sl.Add('Country=USA');

TMSFMXRichEditor1.Merge(sl);

sl.Free;
pic.Free;
```

To undo the merge operation (and have the document ready for a new merge operation), simply call TMSFMXRichEditor1.UnMerge; after the merge operation.

Using accompanying toolbars

TTMSFMXRichEditor comes with 2 ready-to-use toolbars that enable to quickly create user-interfaces for manipulating the formatted text without writing code. To start using the toolbars, simple drop one of the toolbars on either a TTMSFMXDockPanel or directly on the form.

TTMSFMXRichEditorEditToolBar, TTMSFMXRichEditorFormatToolBar

These are 2 toolbars designed to be used in combination with a TTMSFMXDockPanel. The toolbars are divided in functions for Open/Save/Clipboard/Undo/Redo with the TTMSFMXRichEditorEditToolBar, changing font characteristics, alignment, bullets, indents, colors and inserting images, hyperlinks, special characters with the TTMSFMXRichEditorFormatToolbar.



Importing & exporting in rich text

TTMSFMXRichEditor comes with a component to allow to import or export its content in rich text (.RTF) files.
Performing such export or import is easy. Drop a TTMSFMXRichEditorRTFIO component on the form and connect the TTMSFMXRichEditor to this non-visual component's RichEditor property.

Export

Simply call:

```
TMSFMXRichEditorRTFIO.Save(FileName);
```

Import

Simply call:

```
TMSFMXRichEditorRTFIO.Load(FileName);
```

Importing & exporting in HTML format

TTMSFMXRichEditor comes with a component to allow to export its content in HTML (.HTML) files. It is also possible to import from files that use a HTML subset (mini HTML) described here: http://www.tmssoftware.com/site/minihtml.asp

Performing such export or import is easy. Drop a TTMSFMXRichEditorHTMLIO component on the form and connect the TTMSFMXRichEditor to this non-visual component's RichEditor property.

Export

Simply call:

```
TMSFMXRichEditorHTMLIO.Save(FileName);
```

Notice that for HTML export, the default behaviour is that all images used in the document are exported as separate linked image files in the same folder where the .HTML file is generated. If it is preferred that images are generated in a different folder, use the 2nd default parameter ImagePath:

```
TMSFMXRichEditorHTMLIO.Save(FileName, ImagePath);
```

Import

This is limited to mini HTML files and import is done via the non-visual component TTMSFMXRichEditorMiniHTMLIO. In the same way as TTMSFMXRichEditorHTMLIO, assign the TTMSFMXRichEditor instance via TTMSFMXRichEditorMiniHTMLIO.RichEditor. The component provides the following overloads to import from HTML:

```
procedure Load(HtmlValue: string; Pictures: TTMSFMXBitmapContainer);
overload;
procedure Load(FileName: string; Encoding: TEncoding = nil); overload;
procedure Load(AStream: TStream; Encoding: TEncoding = nil); overload;
```

This way, it can import from a simple HTML formatted string, a file with HTML formatted text or a stream. In the case of loading from a HTML formatting string, 1 extra parameter Pictures can be used as a container for referenced images in the HTML formatted string.

Finally, one more helper method is available in TTMSFMXRichEditorMiniHTMLIO:

```
procedure Insert(AHtmlValue: string);
```

This inserts the formatted text from a HTML formatted string at caret position in the TTMSFMXRichEditor.

Exporting to PDF

The TMS Component Pack also contains a component for exporting the TTMSFMXRichEditor content to PDF file.
Drop a TTMSFMXRichEditorPDFCreatorIO component on the form and connect the TTMSFMXRichEditor to this non-visual component's RichEditor property.

Then simply call:

```
TMSFMXRichEditorPDFCreatorIO.Save(FileName);
```

TTMSFMXRichEditorPDFCreatorIO comes with settings for header and footer as well as metadata. Header and footer can as such be optionally generated for the PDF file independently from the TTMSFMXRichEditor content.

Import or export to mini-HTML

With the component TTMSFMXRichEditorMiniHTMLIO, it is possible to read or write the contents of the TTMSFMXRichEditor in mini-HTML format. Mini-HTML is a subset of HTML and is described at: http://www.tmssoftware.com/site/minihtml.asp

To use TTMSFMXRichEditorMiniHTML to read or write its contents in HTML, drop TTMSFMXRichEditorMiniHTML on the form and connect the TTMSFMXRichEditor instance to TTMSFMXRichEditorMiniHTMLIO.RichEditor.

Call TTMSFMXRichEditorMiniHTMLIO.Load(AFileName: string; AEncoding: TEncoding = nil) to load the content from a HTML file.

Call TTMSFMXRichEditorMiniHTMLIO.Save(AFileName: string) to save the content to a HTML file.

In addition to saving to file, it is also possible to save to a stream or get the content as HTML:

TTMSFMXRichEditorMiniHTMLIO.Save(AStream: TStream; AEncoding: TEncoding = nil) : saves the content in HTML format to stream

TTMSFMXRichEditorMiniHTMLIO.AsString: string : returns the content in HTML format as string

In addition to loading from file, it is also possible to get the content from a stream or a HTML formatted string:

TTMSFMXRichEditorMiniHTMLIO.Load(AStream: TStream);

Loads the content from a stream containing the HTML formatted text.

TTMSFMXRichEditorMiniHTMLIO.LoadFromString(AHtmlValue: string; APictures: TTMSFMXBitmapContainer = nil);

HTMLValue contains the content as HTML formatted string. Optionally, for passing pictures, a TTMSFMXBitmapContainer can be used in case the HTML formatted string references pictures in a TTMSFMXBitmapContainer.

TTMSFMXTouchKeyboard / TTMSFMXPopupTouchKeyboard

Description
TTMSFMXTouchKeyboard is a compact light-weight virtual keyboard for desktop applications on Mac and Windows, which includes typical keyboards as QWERTY, QWERTZ, AZERTY, DVORAK, Cellphone and Numeric. The possibility exists to use the keyboard as a popup.
The keyboard can be fully customized in order and style.

Properties & Events
TTMSFMXTouchKeyboard

AutoCapsDisplay	If turned on, the keyboard will mimic the capslock of the normal keyboard
BorderRounding	Sets the roundings of the keys
HighlightCaps	Sets the highlightcolor for the available capital keys
HighlightAltGr	Sets the highlightcolor for the available alt keys
BitmapContainer	A bitmap container for image storage
KeyboardType	Sets the prefered keyboard type
Keys	The key collection
OnKeyClick	When a key was clicked
OnKeyDown	When the key down starts
OnDrawKey	When painting the key

TTMSFMXPopupTouchKeyboard

AutoCapsDisplay	If turned on, the keyboard will mimic the capslock of the normal keyboard
HighlightCaps	Sets the highlightcolor for the available capital keys
HighlightAltGr	Sets the highlightcolor for the available alt keys
KeyboardType	Sets the prefered keyboard type
OnClose	When the popup closes
OnKeyboardCreated	When the keyboard was created

TTMSFMXTouchKeyItem

X	The actual x position of the key
Y	The actual y position of the key
Caption	The normal key caption
ShiftCaption	The key caption while pressing shift
AltGrCaption	The key caption while pressing alt
KeyValue	The key value in normal state
ShiftKeyValue	The key value while pressing shift
AltGrKeyValue	The key value while pressing alt
PictureDownState	The picture shown when the key is being pressed
PictureNormalState	The picture in normal state
SpecialKey	If this key is special, for example the ctrl key
BorderColor	The border color in normal state
borderColorDown	The border color when the key is being pressed
Color	The color in normal state
ColorDown	The color when the key is being pressed
TextColor	The text color

TextColorDown	The text color when the key is being pressed
ImageName	The name of the image, corresponding with the assigned bitmapcontainer
ShortCut	The keys' shortcut
Hint	The hint that's shown when hovering over a key

Methods

TTMSFMXTouchKeyboard

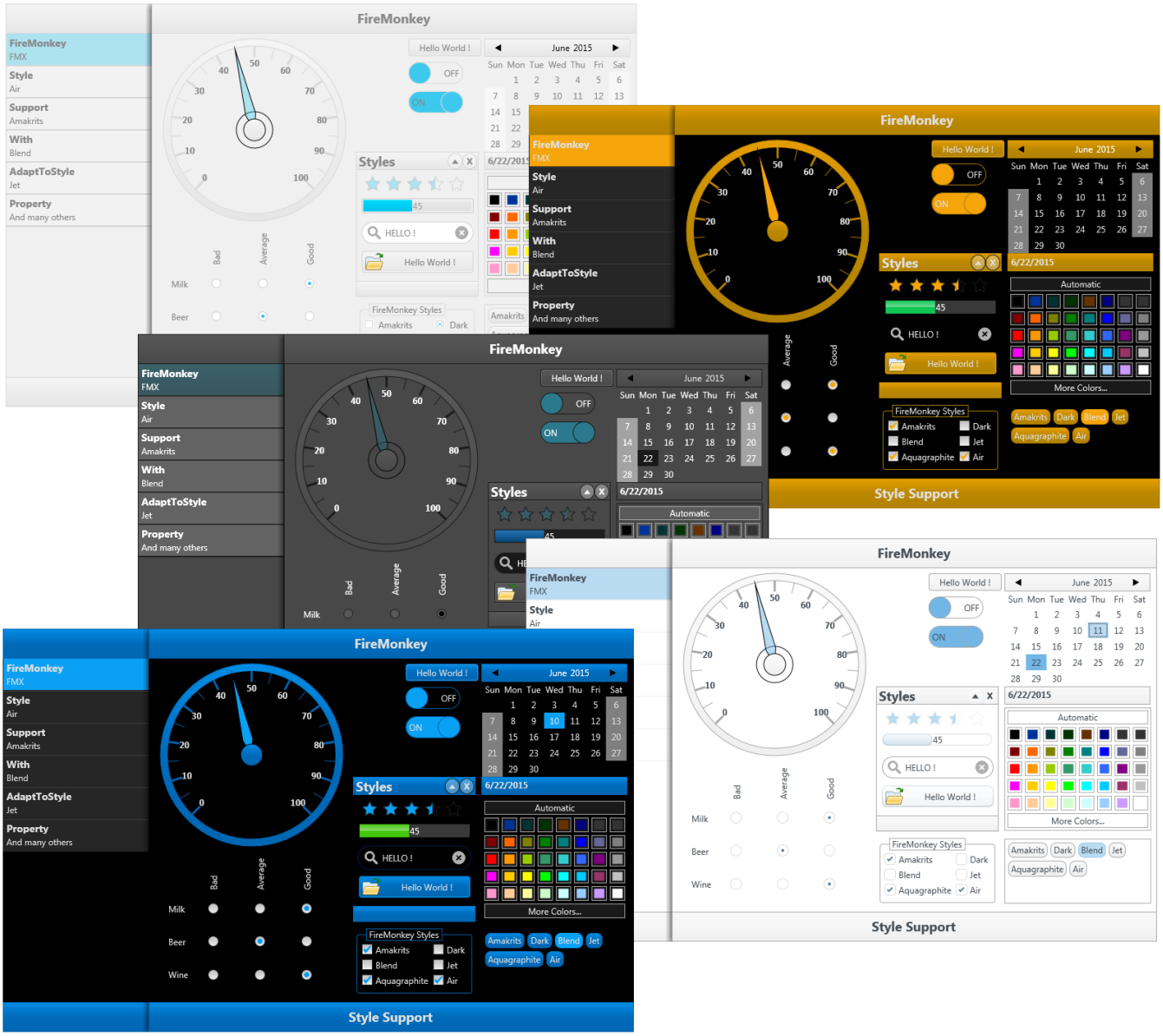
SaveKeybdLayout	Saves the keyboard to the specified file name
LoadKeybdLayout	Loads the keyboard from the specified file name

TTMSFMXPopupTouchKeyboard

ShowKeyboard	Shows the keyboard on the cursors position

Styling

The TMS Pack for FireMonkey supports FireMonkey Styles. Simply put a StyleBook on the form and load one of the default or premium FireMonkey Styles. After assigning the StyleBook to the form, the AdaptToStyle property of the selected component will then automatically adapt to the style loaded in the StyleBook. Below is a sample of styles that are applied to the TMS Pack for FireMonkey components.



Export to PDF

The TMS Pack for FireMonkey contains several components for exporting the components content to PDF file. This includes: TTMSFMXGridPDFCreatorIO, TTMSFMXRichEditorPDFCreatorIO, TTMSFMXMemoPDFCreatorIO.
Drop a TTMSFMXGrid, TTMSFMXRichEditor, TTMSFMXMemo on the form and its corresponding *PDFCreatorIO non-visual component on the form and connect it to respectively the .Grid, .RichEditor, .Memo property of the non-visual component.

To get PDF generated, simply call the .Save() function.

The *PDFCreatorIO components come with settings for PDF file header and footer as well as PDF metadata. Header and footer can as such be optionally generated for the PDF file independently from the components content.

Properties
Header & Footer:

PageNumberFont	Sets the font for the page number
PageNumberHAlign	Sets the page number horizontal alignment
PageNumberVAlign	Sets the page number vertical alignment
PageNumberVisible	Makes the page number (in)visible
Text	Sets the Text
TextFont	Sets the font for the text
TextHAlign	Sets the horizontal alignment for the text
TextVAlign	Sets the vertical alignment for the text

Metadata

Company	The company name
CreateDate	Date of PDF creation
Creator	Name of the person
CreatorTool	The program name
Producer	Name of the producer
Title	The PDF title

PageLayout:
Sets the layout of the document in portrait or landscape

Component specific property: Grid, RichEditor or Memo
This will be the instance of the component to link with and export to PDF.

Saving the document:
A single line of code converts the TTMSFMXGrid, TTMSFMXRichEditor, TTMSFMXMemo component content to a PDF document.
This can be done by code:

```
TTMSFMXRichEditorPDFCreatorIO.Save(FileName);
```

where in this case the component to export for would be a TTMSFMXRichEditor instance.

Please take in account that the .Save() method's path parameter is platform specific!

Windows, Mac OS-X: Document can be saved anywhere where OS file system permits
IOS: The document can only be saved in the private app's documents folder
Android: The document can be saved on the device or SD-card if the app has sufficient permissions

TMS Mini HTML rendering engine

Another core technology used among many components is a small fast & lightweight HTML rendering engine. This engine implements a subset of the HTML standard to display formatted text. It supports following tags :

B : Bold tag
<B> : start bold text
</B> : end bold text

Example : This is a <B>test</B>

U : Underline tag
<U> : start underlined text
</U> : end underlined text

Example : This is a <U>test</U>

I : Italic tag
<I> : start italic text
</I> : end italic text

Example : This is a <I>test</I>

S : Strikeout tag
<S> : start strike-through text
</S> : end strike-through text

Example : This is a <S>test</S>

A : anchor tag
<A href="value"> : text after tag is an anchor. The 'value' after the href identifier is the anchor. This can be an URL (with ftp,http,mailto,file identifier) or any text.
If the value is an URL, the shellexecute function is called, otherwise, the anchor value can be found in the OnAnchorClick event </A> : end of anchor

Examples : This is a <A href= "mailto:myemail@mail.com ">test</A>
This is a <A href="http://www.tmssoftware.com">test</A>
This is a <A href="somevalue">test</A>

FONT : font specifier tag
<FONT face='facevalue' size='sizevalue' color='colorvalue' bgcolor='colorvalue'> : specifies font of text after tag.
with

- face : name of the font
- size : HTML style size if smaller than 5, otherwise pointsize of the font
- color : font color with either hexidecimal color specification or color constant name, ie claRed,claYellow,claWhite ... etc
- bgcolor : background color with either hexidecimal color specification or color constant name </FONT> : ends font setting

Examples : This is a <FONT face="Arial" size="12" color="clared">test</FONT>
This is a <FONT face="Arial" size="12" color="#FF0000">test</FONT>

P : paragraph
<P align="alignvalue" [bgcolor="colorvalue"] [bgcolorto="colorvalue"]> : starts a new paragraph, with left, right or center alignment. The paragraph background color is set by the optional bgcolor parameter. If bgcolor and bgcolorto are specified,
a gradient is displayed ranging from begin to end color.
</P> : end of paragraph

Example : <P align="right">This is a test</P>
Example : <P align="center">This is a test</P>
Example : <P align="left" bgcolor="#ff0000">This has a red background</P>
Example : <P align="right" bgcolor="claYellow">This has a yellow background</P>
Example : <P align="right" bgcolor="claYellow" bgcolorto="clared">This has a gradient background</P>*

HR : horizontal line
<HR> : inserts linebreak with horizontal line

BR : linebreak

 : inserts a linebreak

BODY : body color / background specifier
<BODY bgcolor="colorvalue" [bgcolorto="colorvalue"] [dir="v|h"] background="imagefile specifier"> : sets the background color of the HTML text or the background bitmap file

Example : <BODY bgcolor="claYellow"> : sets background color to yellow
<BODY background="file://c:\test.bmp"> : sets tiled background to file test.bmp
<BODY bgcolor="claYellow" bgcolorto="claWhite" dir="v"> : sets a vertical gradient from yellow to white

IND : indent tag
This is not part of the standard HTML tags but can be used to easily create multicolumn text
<IND x="indent"> : indents with "indent" pixels

Example :
This will be <IND x="75">indented 75 pixels.

IMG : image tag
<IMG src="specifier:name" [align="specifier"] [width="width"] [height="height"] [alt="specifier:name"] > : inserts an image at the location

specifier can be: name of image in a BitmapContainer

Optionally, an alignment tag can be included. If no alignment is included, the text alignment with respect to the image is bottom. Other possibilities are: align="top" and align="middle"

The width & height to render the image can be specified as well. If the image is embedded in anchor tags, a different image can be displayed when the mouse is in the image area through the Alt attribute.

Examples :
This is an image <IMG src="name">

SUB : subscript tag

<SUB> : start subscript text
</SUB> : end subscript text

Example : This is ⁹/₁₆ looks like 9/16

SUP : superscript tag
<SUP> : start superscript text
</SUP> : end superscript text

UL : list tag
<UL> : start unordered list tag
</UL> : end unordered list

Example : <UL>
<LI>List item 1
<LI>List item 2
<UL>
<LI> Sub list item A
<LI> Sub list item B
</UL>
<LI>List item 3
</UL>

LI : list item
<LI [type="specifier"] [color="color"] [name="imagename"]>: new list item specifier can be "square", "circle" or "image" bullet. Color sets the color of the square or circle bullet. Imagename sets the PictureContainer image name for image to use as bullet

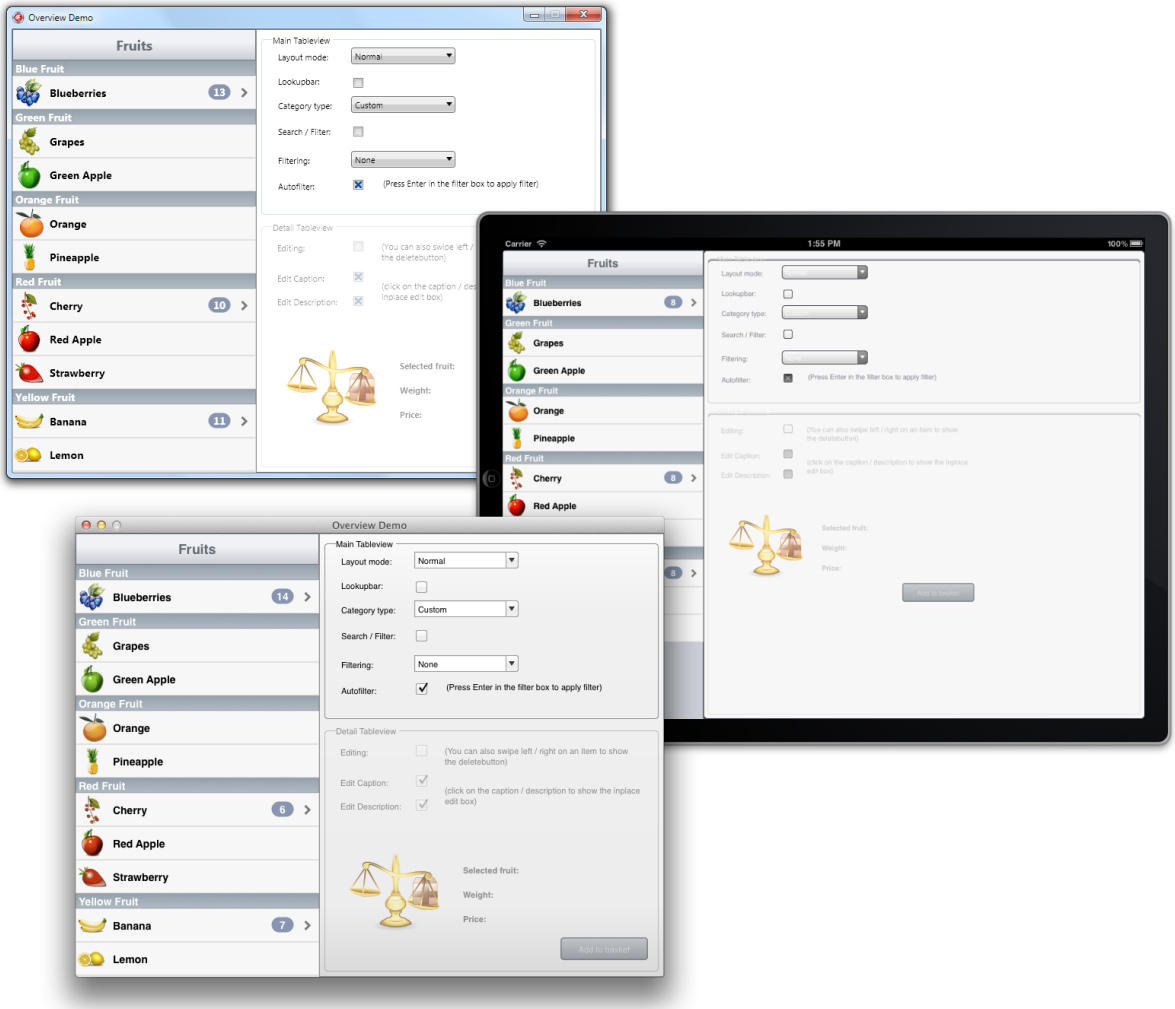
SHAD : text with shadow
<SHAD> : start text with shadow
</SHAD> : end text with shadow

Z : hidden text
<Z> : start hidden text
</Z> : end hidden text

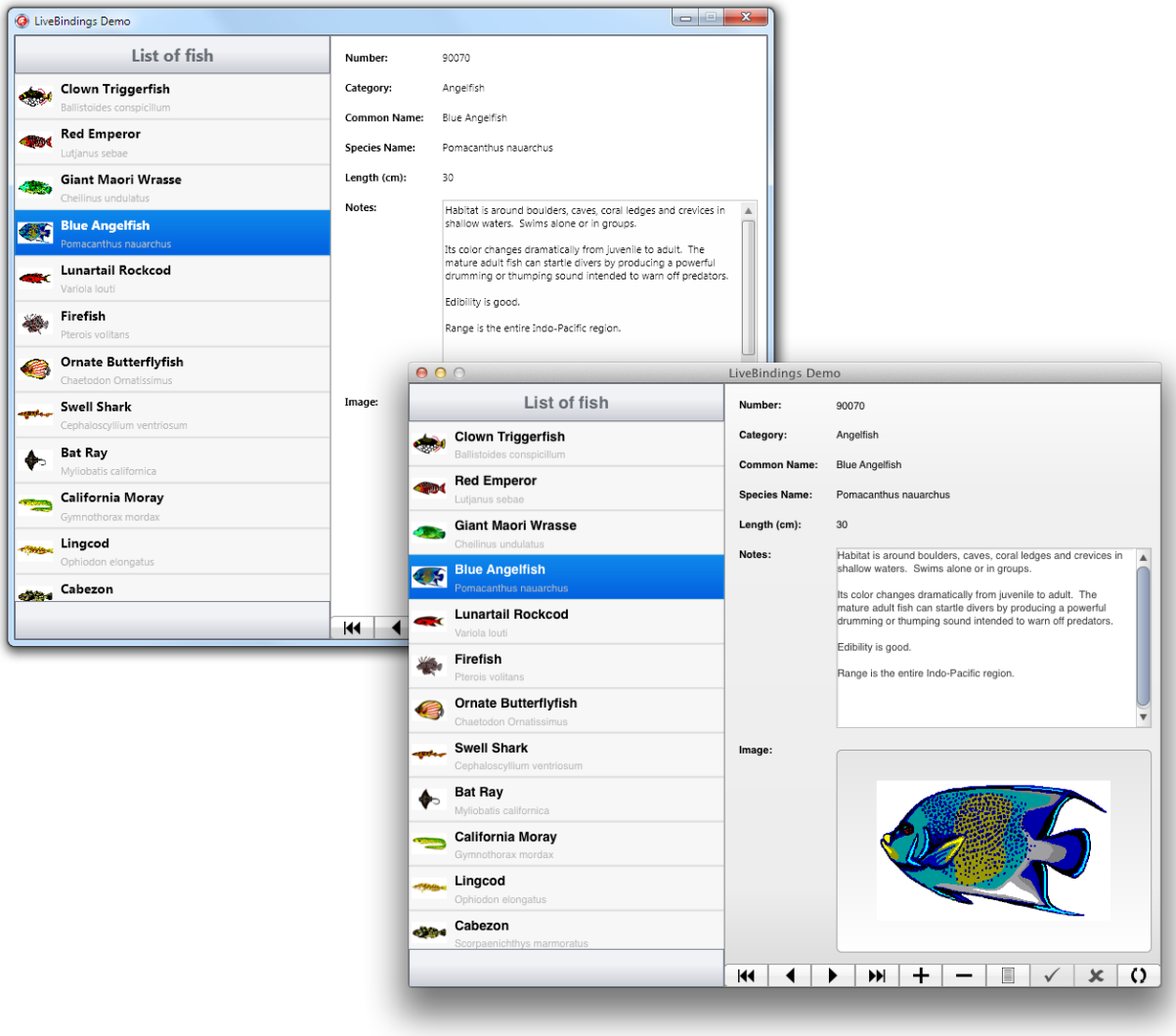
Special characters
Following standard HTML special characters are supported :
< : less than : <
> : greater than : >
& : &
" : "
  : non breaking space
™ : trademark symbol
€ : euro symbol
§ : section symbol
© : copyright symbol
¶ : paragraph symbol

Samples

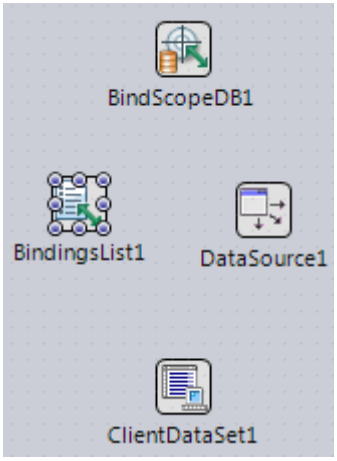
TMS TableView Overview Demo



TMS TableView LiveBindings Demo 1 & 2



These demos are LiveBindings samples, specifically designed in combination with the TMSTableView. The demos load a ClientDataSet with a biolife.xml sample data file. The ClientDataSet is then connected, in combination with a DataSource, to a BindScope, which is needed to bind data to the ListBindings component.



The demos makes use of a TTMSFMXBindDBTableViewLink that is created and registered specifically for the TableView component. When opening the editor of the BindingsList you will see various bindings to different elements that are placed on the right side of the TableView. All these elements have a binding to a specific field and will update when navigating through the list or with the BindNavigator component.

Name	Description
DBLinkImageControl1Graphic1	Link control "ImageControl1" to source "BindScopeDB1, Graphic"
DBLinkLabel2SpeciesNo1	Link control "Label8" to source "BindScopeDB1, Species No"
DBLinkLabel12Category1	Link control "Label12" to source "BindScopeDB1, Category"
DBLinkLabel10Common_Name1	Link control "Label10" to source "BindScopeDB1, Common_Name"
DBLinkLabel6SpeciesName1	Link control "Label6" to source "BindScopeDB1, Species Name"
DBLinkLabel4Lengthcm1	Link control "Label4" to source "BindScopeDB1, Length (cm)"
DBLinkMemo1Notes1	Link control "Memo1" to source "BindScopeDB1, Notes"
TMSFMXBindDBTableViewLink1	Link grid control "TMSFMXTableView1" to source "BindScopeDB1"

Selecting the TMSFMXBindDBTableViewLink1 component will display its properties in the object inspector.

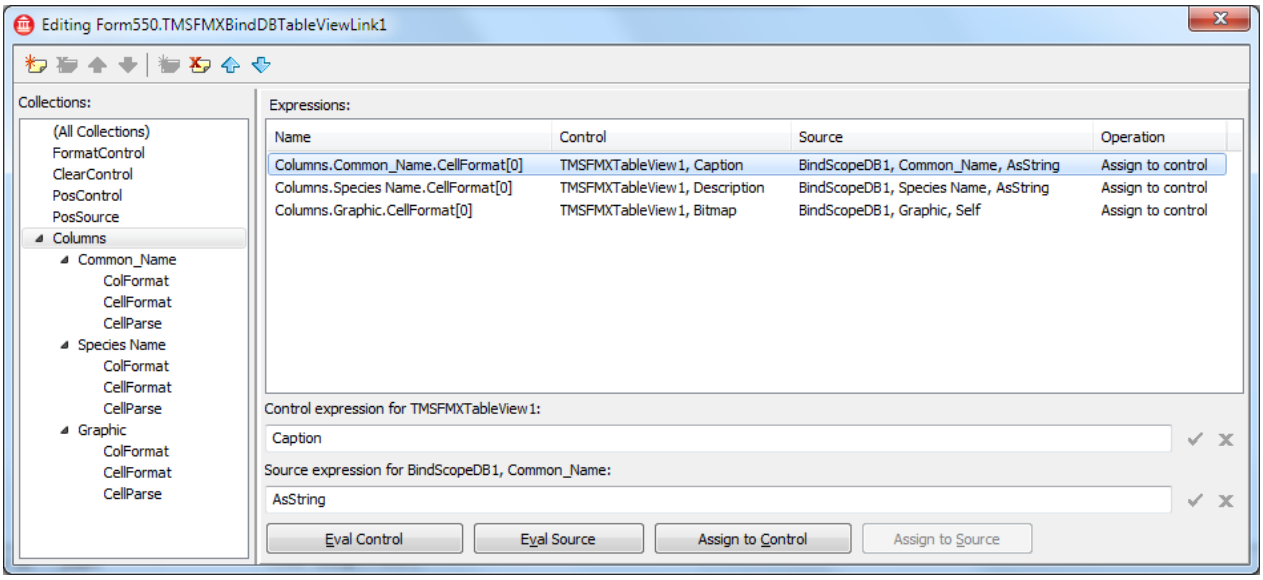
TMSFMXBindDBTableViewLink1 TTMSFMXBindDBTableViewLink

Properties | Events

AutoActivate	True
AutoFill	True
BufferCount	-1
Category	**DB TableView Links**
ClearControlExpress	**(TExpressions)**
ColumnExpressions	**(TColumnLinkExpressions)**
ControlComponent	**TMSFMXTableView1**
FormatControlExpre	**(TExpressions)**
Name	TMSFMXBindDBTableViewLink1
PosControlExpressic	**(TExpressions)**
PosSourceExpressio	**(TExpressions)**
SourceComponent	**BindScopeDB1**
Tag	0

As with the TrackBar sample in the LiveBindings chapter this link has a SourceComponent and a ControlComponent. The difference between this link and the link in the TrackBar sample is, that the TMSFMXBindDBTableViewLink component is able to bind multiple fields to multiple elements in the TableView.

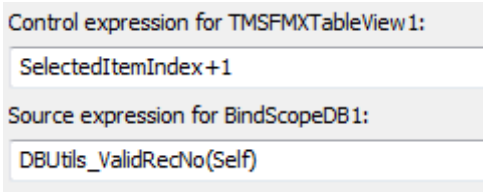
In these samples you will notice that there is binding to the Caption, Description and the Bitmap in the TableView. To bind data to these elements, an expression must be added per element to the ColumnExpressions collection. Double-clicking on the ColumnExpressions property opens the expressions editor.



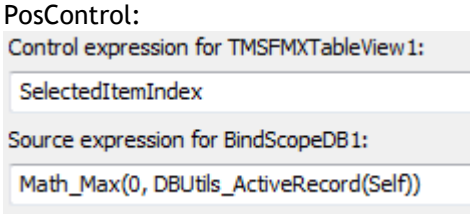
For the caption and the Description, the binding has been added to a string field in the DataBase. To return the value for the current record, the AsString function must be used in the Source expression. To bind the graphic, the source expression is Self. To know exactly which expression you must use, you can click on Eval Source which will tell you the type of the data.

When starting the application, the list will load the data and display the caption, description and bitmap. When clicking on the list, the navigation in the dataset is automatically handled. This is due to the initialization of the TTMSFMXBindDBTableViewLink that has already taken care of this functionality and has stored the correct values in the PosSource and PosControl expressions.

PosSource:



PosControl:



By default the tableview has properties for the Caption, Description and Bitmap. This can be expanded to link to multiple fields in the database. After installing the package you will also notice a TTMSFMXTableViewEx. This implements an extended item class that introduces a new property: Extra.
This property is then linked to an extra element that manually needs to be added through the stylebook editor with the correct name. This is a sample, so this can easily be expanded with multiple properties for multiple elements.

Below is the code of the extended list with the extended item property.

```
unit FMX.TMSTableViewEx;

interface

uses
  Classes, FMX.TMSTableView, FMX.Objects;


type
  TTMSFMXTableViewItemEx = class(TTMSFMXTableViewItem)
  private
    FExtra: string;
    procedure SetExtra(const Value: string);
  protected
    procedure UpdateExtra;
    procedure ApplyEx; override;
  public
    function ShapeExtra: TText;
  published
    property Extra: string read FExtra write SetExtra;
  end;

  TTMSFMXTableViewItemsEx = class(TTMSFMXTableViewItems)
  public
    function CreateItemClass: TCollectionItemClass; override;
  end;

  TTMSFMXTableViewEx = class(TTMSFMXTableView)
  protected
    function GetClassStyleName: String; override;
  public
    function CreateItems: TTMSFMXTableViewItems; override;
  end;


implementation

{ TTMSFMXTableViewItemsEx }

function TTMSFMXTableViewItemsEx.CreateItemClass: TCollectionItemClass;
begin
  Result := TTMSFMXTableViewItemEx;
end;

{ TTMSFMXTableViewEx }

function TTMSFMXTableViewEx.CreateItems: TTMSFMXTableViewItems;
```

```
begin
  Result := TTMSFMXTableViewItemsEx.Create(Self);
end;

function TTMSFMXTableViewEx.GetClassStyleName: String;
begin
  Result := ClassParent.ClassName + 'style';
  Delete(Result, 1, 1);
end;

{ TTMSFMXTableViewItemEx }

procedure TTMSFMXTableViewItemEx.ApplyEx;
begin
  inherited;
  UpdateExtra;
end;

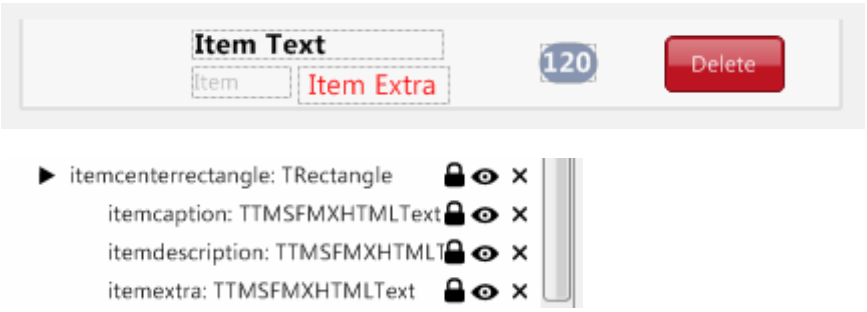
procedure TTMSFMXTableViewItemEx.SetExtra(const Value: string);
begin
  if FExtra <> Value then
  begin
    FExtra := Value;
    if TableView.GetUpdateCount = 0 then
      UpdateExtra;
  end;
end;

function TTMSFMXTableViewItemEx.ShapeExtra: TText;
var
  shp: TTMSFMXTableViewItemShape;
begin
  Result := nil;
  shp := Shape;
  if Assigned(shp) then
    Result := shp.FindStyleResource('itemextra') as TText;
end;

procedure TTMSFMXTableViewItemEx.UpdateExtra;
var
  shp: TText;
begin
  shp := ShapeExtra;
  if Assigned(shp) then
    shp.Text := Extra;
end;

end.
```

When dropping this TTMSFMXTableViewEx on the form, you will notice that it has the same appearance as the normal TTMSFMXTableview. This is because the extra element needs to be added to the list. Edit the custom style and add a new TTMSFMXHTMLText to the itemcenterrectangle so it looks like the item appearance below:

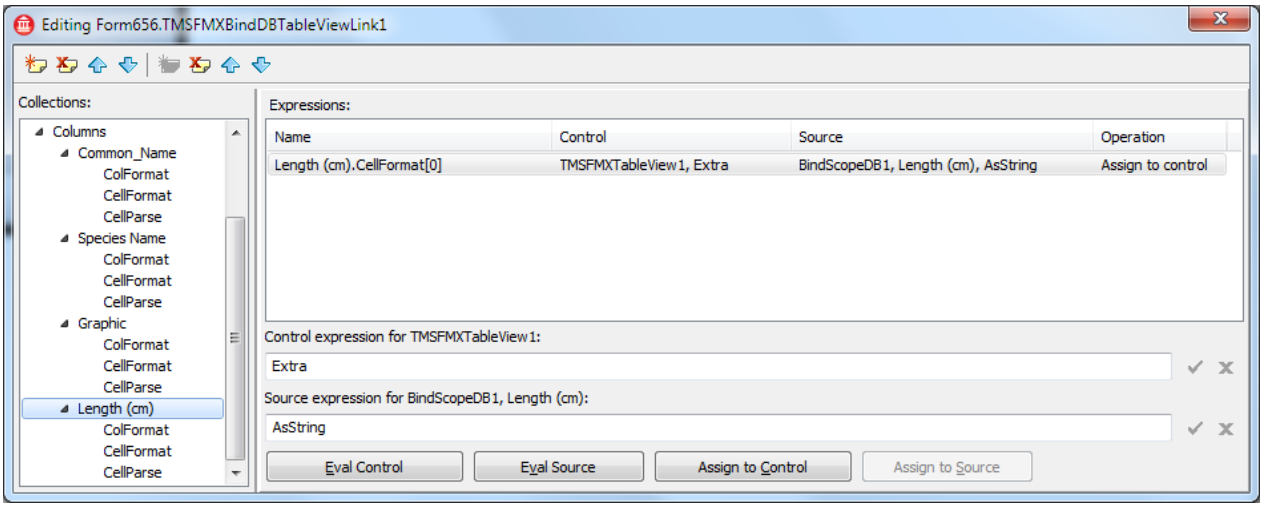


Set the stylename of that item to “itemextra” since that is the name that is used in the TTMSFMXTableViewEx component to access this element.

In the second demo the list is loaded with data from the biolife.xml and displaying an additional field in the item:

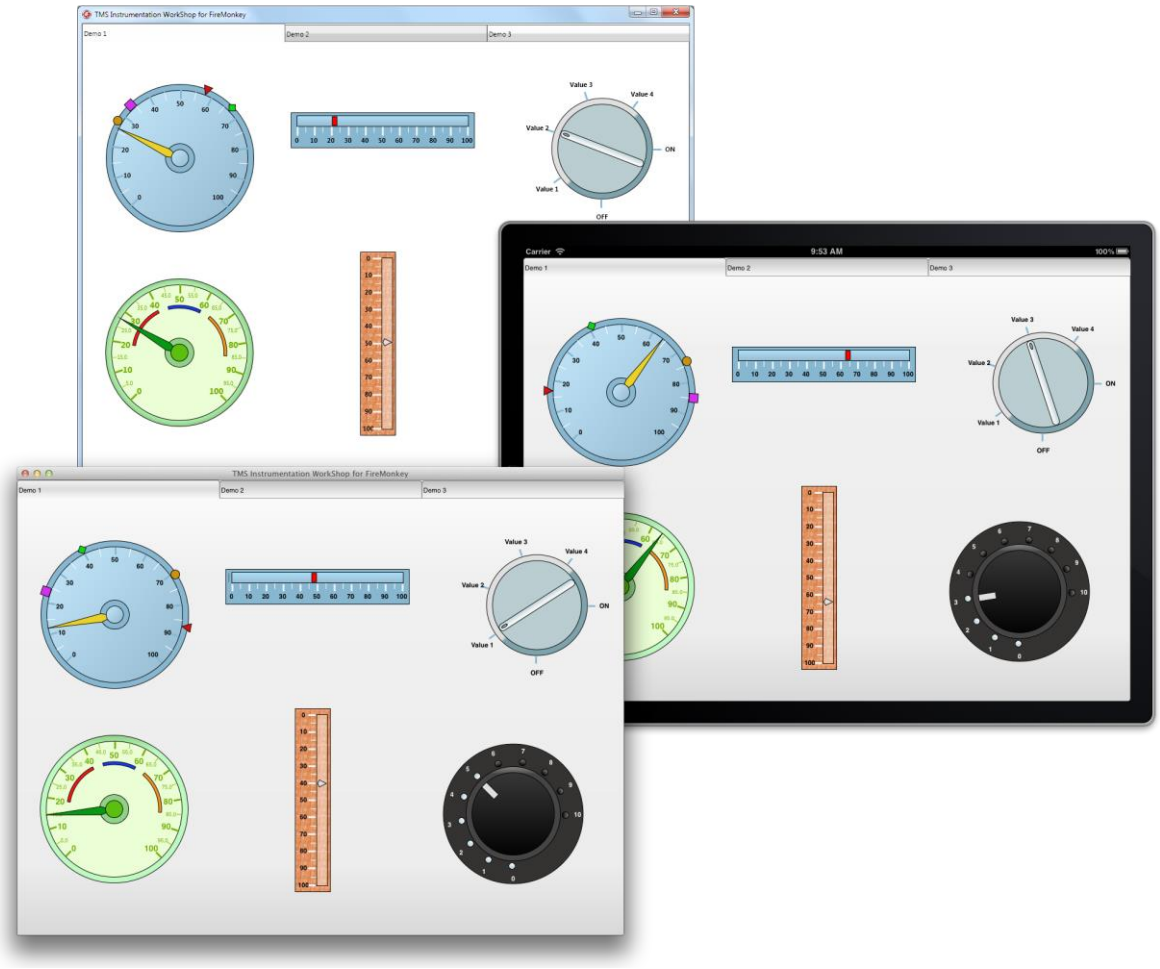


If you open the BindingsList editor and open the BindDBTableViewLink expression editor you will notice an extra column is now added and linked to the Extra property that is available in this extended TTMSFMXTableViewEx component.

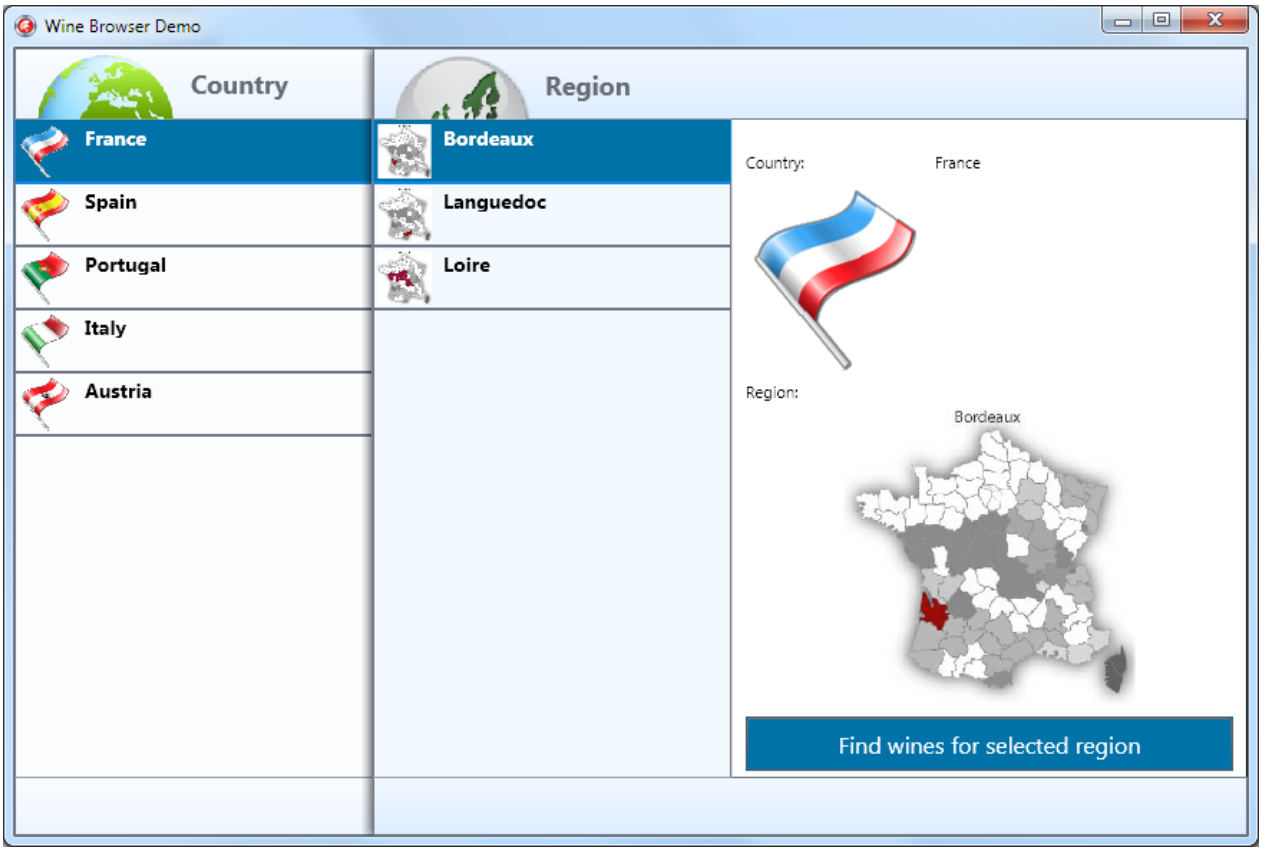


This way of introducing properties can be done for other properties and can be combined with various elements to allow displaying all fields in the tableview component.

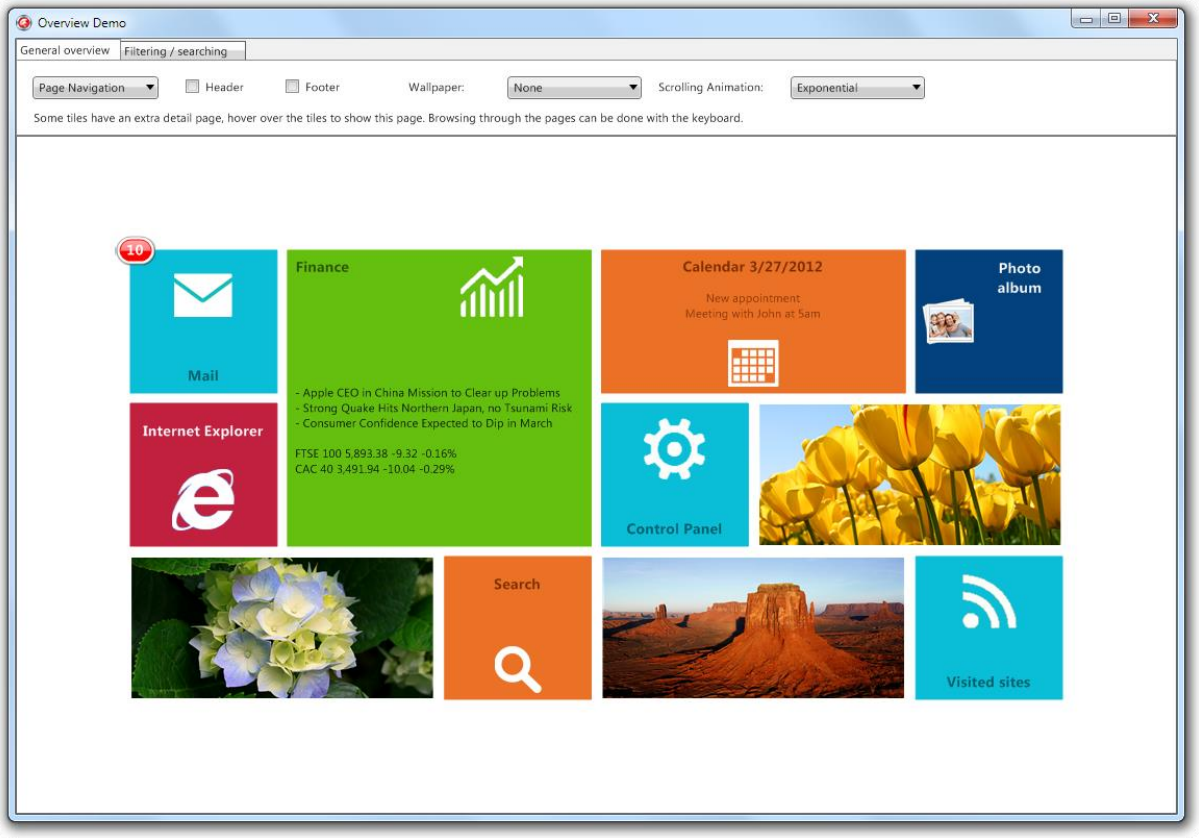
TMS Instrumentation WorkShop Demo



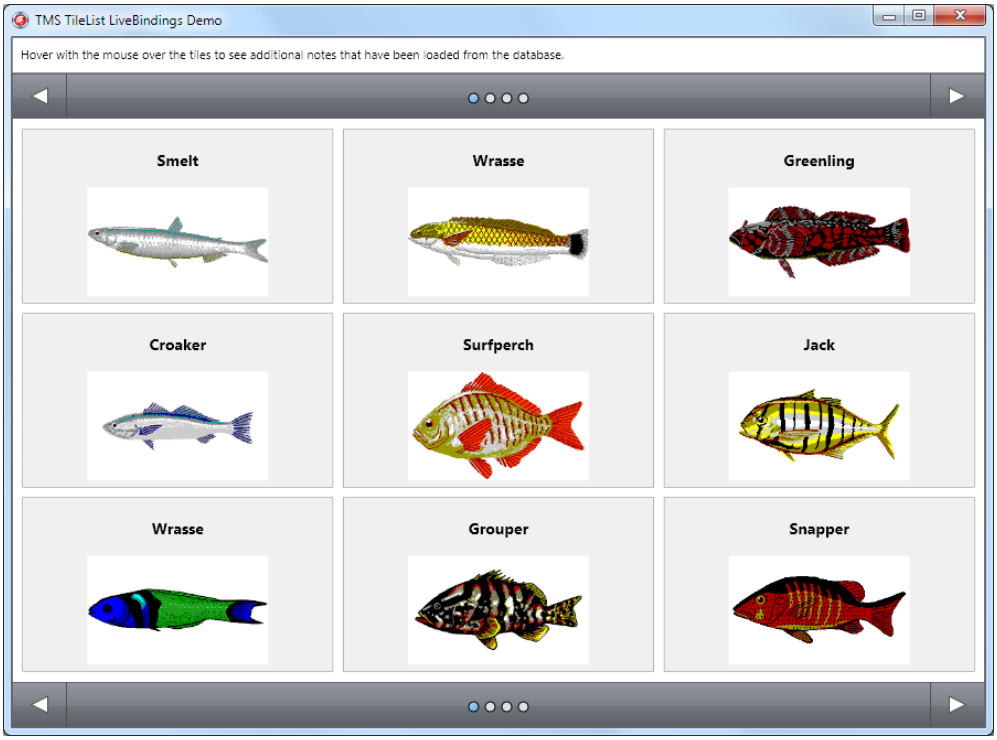
TMS PageSlider Demo



TMS TileList Demo



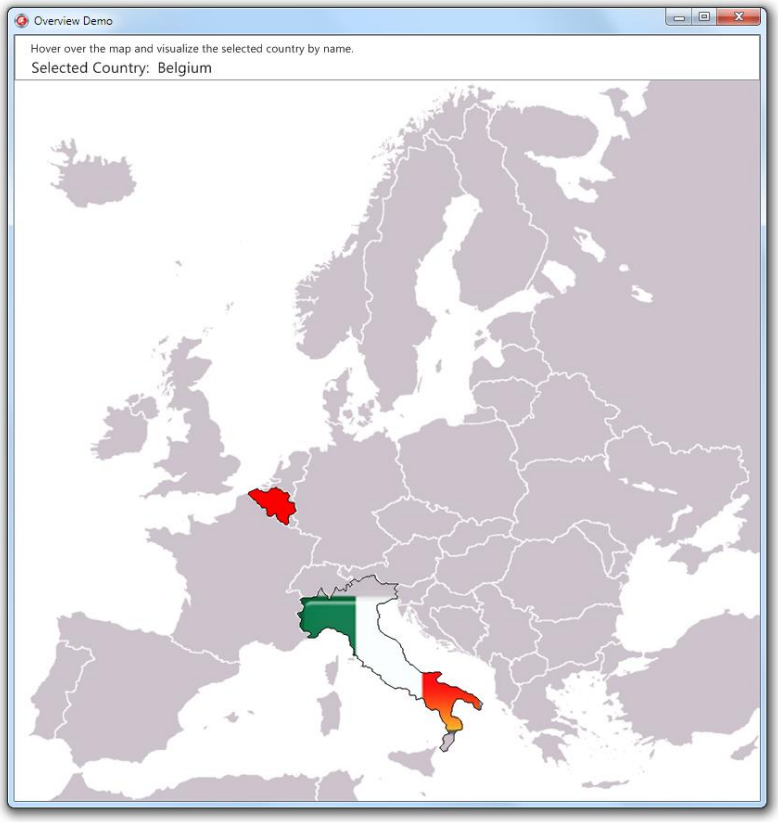
TMS TileList LiveBindings Demo



This demo is a LiveBindings sample, specifically designed in combination with the TMSTileList. The demo loads a ClientDataSet with a biolife.xml sample data file. The ClientDataSet is then connected, in combination with a DataSource, to a BindScope, which is needed to bind data to the ListBindings component.

More information for LiveBindings and samples can be found at the TMS TableView LiveBindings samples and / or LiveBindings chapter.

TMS HotSpotImage Demo



General FireMonkey component usage guidlines

With the new FireMonkey framework, the methodology to create and use components has dramatically changed. A component now exists of 2 parts.

Visual part

The visual part is stored in a .style file, which is compiled to a .res file through an .rc file. The .rc file is included in the package and must be recompiled whenever a change is made to the .style file. For each component in this set you will find a .style file. In this file, the default layout of the component is stored.

You will notice different elements, basic elements such as an arc, ellipse, rectangle …
The elements combine and define the layout of a control. The basic elements are called shapes, and are already available by default. The TMS Instrumentation Workshop for FireMonkey takes this new way of styling one step further: custom shapes. In several components you will find custom shapes registered and useable in a new application, and used in the component by default.

Each shape or element can have a StyleName, which is used in the non-visual part of the control for interaction. This name is key in the relationship or “style-contract” between style resource and component code.

Non-visual part

The non-visual part of the component interacts with the shapes defined in the .style file. This is a normal .pas unit file as was used for VCL component, yet little to no painting is done in code. As explained above, the visual part is already defined by the style.

The component defined in this unit needs to inherit from the TStyledControl class, which can be styled at designtime. This is the base class for all styleable controls, just like the TCustomControl class was the base class for most controls in the VCL framework.

Naming convention

It is always good practice to handle a consistent naming convention, therefore all .rc, .pas files and .style files should start with the FireMonkey unit scope name “FMX.”, such as the units: FMX.Types, FMX.Dialogs, FMX.Objects …

Inside the style file each element can have a StyleName, which can be used in the non-visual part to address the resource. Make sure each element has a unique StyleName to avoid mistakes when interacting with the component. All combinations of elements must be encapsulated within a rectangle element that is invisible by default (through the Fill.Kind and Stroke.Kind = bkNone), and has the StyleName of the component.

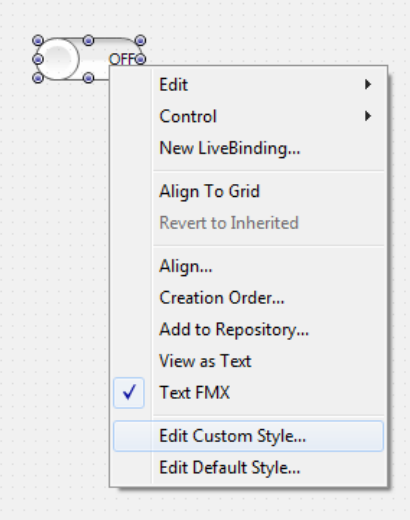
If you have a component named TFMXMyFirstControl, the the StyleName of the rectangle encapsulating all other elements must be set to FMXMyFirstControlStyle. The “T” is removed and “Style” is added.

Styling

Each component inherits from TTMSFMXBaseControl which implements a basic Fill and Stroke, and handles the style resource files that define the default layout of the component. To change the visuals of the component you no longer have corresponding properties in the object inspector.

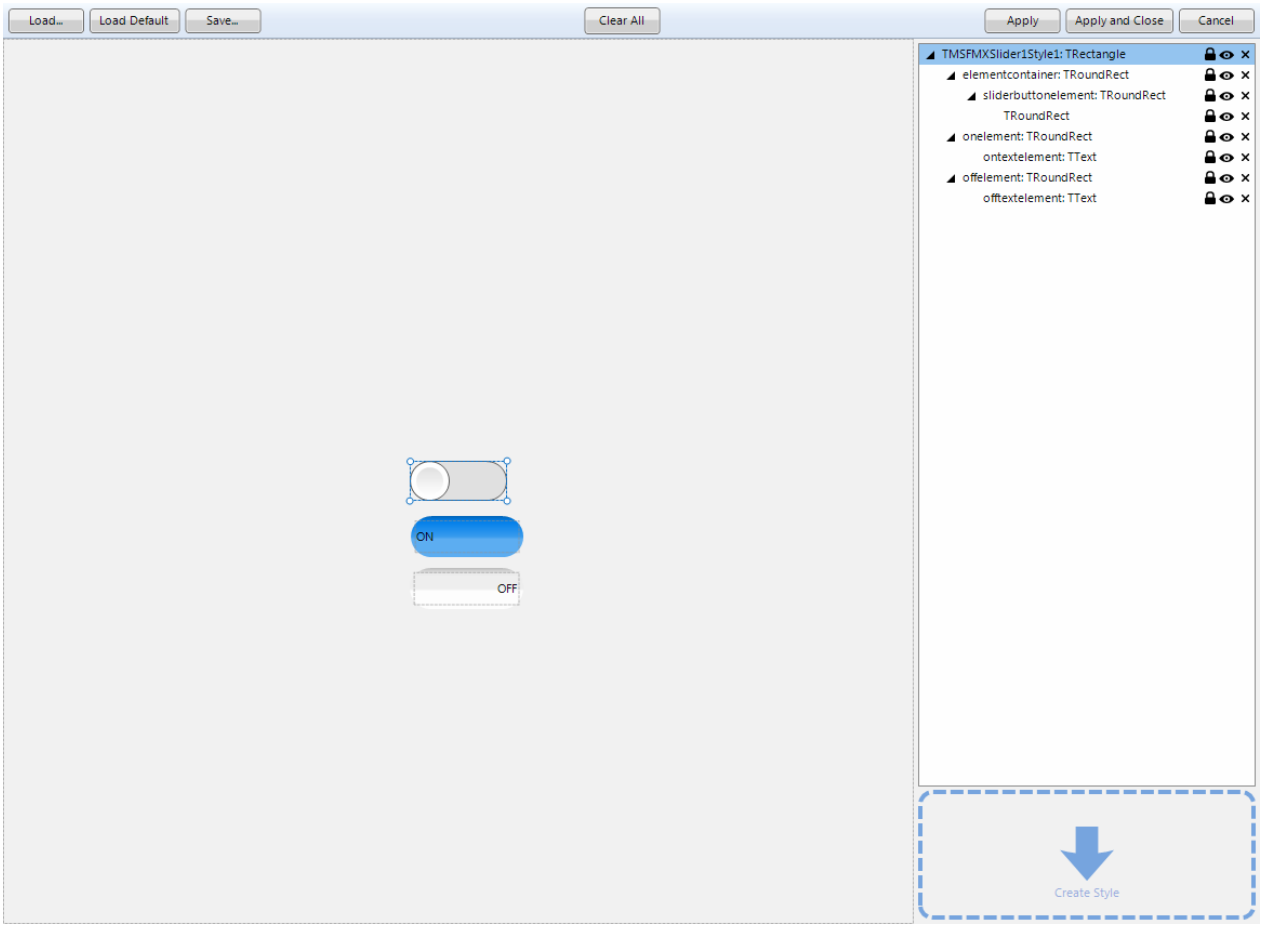
Right-clicking on the component provides two extra menu items that can be used to edit the style of the component.

Clicking either of these items will automatically drop a StyleBook component on the form when there is not yet one available. A StyleBook holds custom and default styles. When the default style is changed, dropping a new component of the same class will automatically get this changed style as defined in the default style.

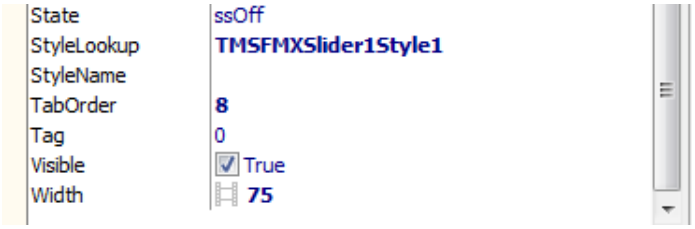


- Edit Custom Style: Clicking on this item starts the IDE style editor and copies the default style of the component. The name of the style is set to the component name on the form and appended with 'Style1'. After changing properties through the editor, the style is then applied to the component. You will notice that the StyleLookUp property is set to the name of the custom style in the stylebook.

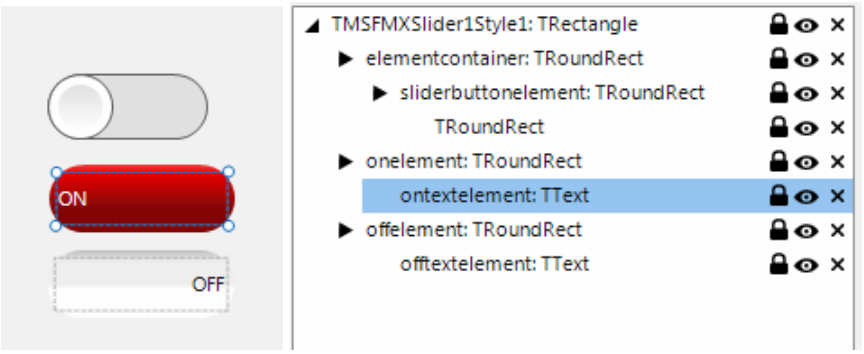
- Edit Default Style: Clicking on this item starts the IDE style editor and uses the default style of the component. As with the Edit Custom Style option, the name of that style is set. The difference between these 2 options is that the default style has a generic name and is applied to all new instances of the component that are dropped on the form. The StyleLoopup property is not set.



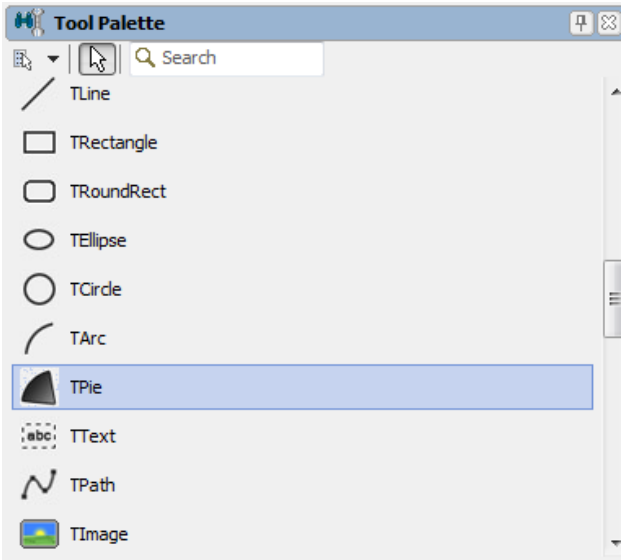
The IDE style editor can be started with these 2 options, or by double-clicking on the StyleBook editor icon on the form. In this example we have a TTMSFMXSlider component that will be altered with a custom style. Notice the TMSFMXSlider1Style1 name that is used for this style. When applying this style, you will also notice the StyleLookup property is set to TMSFMXSlider1Style1.



Each component exists of different styleable elements. Simple click on an element in the editor to change the appearance.



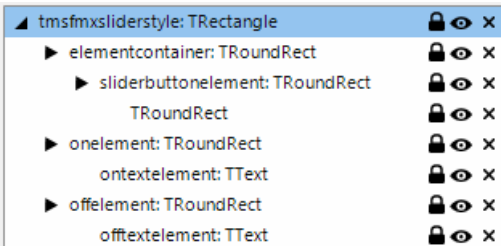
You can also add new elements from the Tool palette.



After applying the Style, the component will have the new custom style.



Dropping a new TTMSFMXSlider component on the form will not adopt this custom style and will have the default style applied. Editing the default style is done in the same way, yet the name of the style differs and each new instance of the TTMSFMXSlider adopts the edited default style.



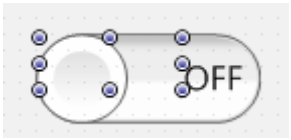
General component properties that do not directly define a visual appearance of the component are still displayed in the Object Inspector. Note though that some properties will affect what is available in the style editor! For example, if a component provides a collection of visible items displayed in the control and it is desirable that the visual appearance of each item can be customized, style elements (shapes) will be dynamically added or removed and be available in the IDE style editor.
In other cases, it is desirable that the appearance for a given type of items in a control is identical. This can be represented as a single style element in the style editor. The component will then internally copy the settings of the style element and apply it to each item displayed in the control.

Components

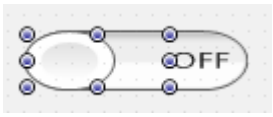
Most of the components in the FireMonkey framework can be scaled and rotated without loss of functionality and quality. As our base control implementation inherits from a base class which supports these features, all of the controls inside the TMS Instrumentation WorkShop set support scaling and rotation.

Scaling: With the Scale property you can specify how large the component must be. The default value of the X and Y property of the Scale is 1. This means that the default component layout is set at one, if you have a component which has 100 pixels width and height dimensions, setting the scale X and Y properties to 1.5 will automatically increase the width and height to 150 pixels. Below are some examples at designtime, which shows the capability of this property.

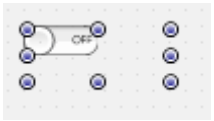
Scale 1.5



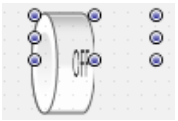
Scale X 1.5 Y 1



Scale 0.5

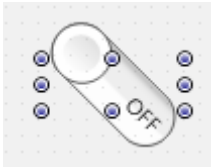


Scale X 0.5 Y 2



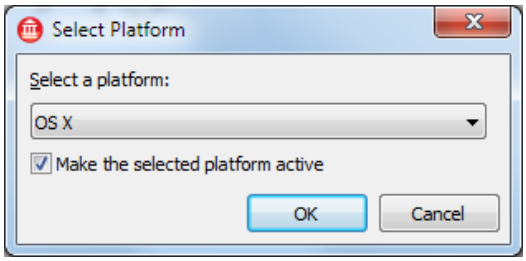
Rotation: The rotation property rotates the component around the center by default, which can be changed with the rotationcenter property. Rotating the component does not limit interaction capabilities and functionality.

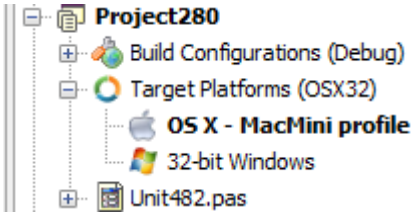
45°



Cross-platform deployment of applications with TMS Pack for FireMonkey components

The FireMonkey framework supports deployment to Windows, Mac OS-X, iOS and Android. Likewise, the TMS Pack for FireMonkey components can be used in applications targeted for these 4 platforms. If you have created a FireMonkey HD project, you can specify in the Project Manager under Target platforms Win32, Win64 or Mac OS-X platforms. Depending on the chosen platform, the compiler will generate a binary for debug or deployment for Windows 32bit, Windows 64bit or Mac OS-X. The app will be automatically compiled with the installed TMS Pack for FireMonkey components.





For iOS and Android, the FireMonkey HD project cannot be used. For this platform, create a new FireMonkey Mobile project. The TMS Pack for FireMonkey components supports both FireMonkey Mobile and FireMonkey HD projects and installs the components on the tool palette for both targets. As such, when choosing a FireMonkey iOS project, the TMS Pack for FireMonkey components will also appear in the IDE component palette.