TMS FNC TableView
DEVELOPERS GUIDE

January 2022

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

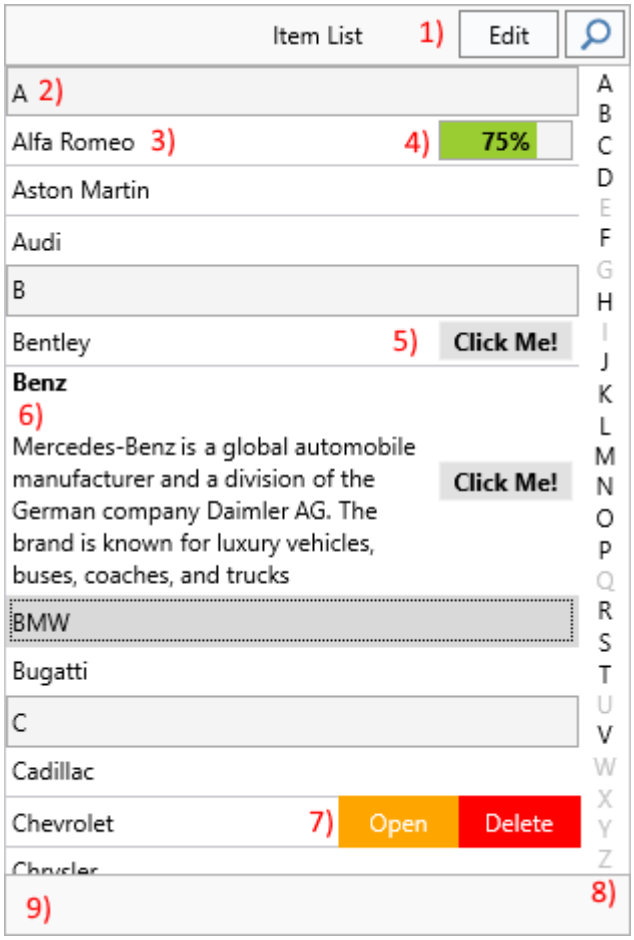
Index

Availability

Supported frameworks and platforms
- VCL Win32/Win64
- FMX Win32/Win64, macOS, iOS, Android, Linux
- LCL Win32/Win64, macOS, iOS, Android, numerous Linux variants including Raspbian
- WEB: Chrome, Edge, Firefox, …

Supported IDE's
- Delphi XE7 and C++ Builder XE7 or newer releases
- Lazarus 1.4.4 with FPC 2.6.4 or newer official releases
- TMS WEB Core for Visual Studio Code 1.3 or newer releases

Hierarchy



1) **Header**: The header is an area that can display text and acts as an area to display the edit, filter and back button. The header is not a separate container, but is part of the TTMSFNCTableView. The header can therefore not be used to align controls. To display controls inside the header you'll need to manually set the position during the resize event.

2) **Section**: The section is an area that indicates the beginning of one or multiple items linked to a specific category. The categories can be alphabetic, alphanumeric or custom.

3) **Item**: The item is an area that can display HTML formatted text as well as additional accessories.

4) **Accessory**: The accessory is an area inside the item that can be used to display additional content such as an image, progressbar, badge or can be used to display custom content as well.

5) **HTML formatted text**: The item is capable of displaying HTML formatted text that is automatically sized according to the length of the text and the available space. Note that more settings regarding item height, wordwrapping and HTML formatted text is possible on item level and item appearance level.

6) **More Options**: The more options collection is a global buttons collection that can be used for each item to display additional actions. Please note that the more options collection is used for each item.

7) **Lookupbar**: The lookupbar displays the categories that has been set using the CategoryType property. If a specific category has no items, the category is in-active and display in a separate font.

8) **Footer**: The footer is an extra area where text can be displayed. The footer is not a separate container, but is part of the TTMSFNCTableView. The footer can therefore not be used to align controls. To display controls inside the footer you'll need to manually set the position during the resize event.

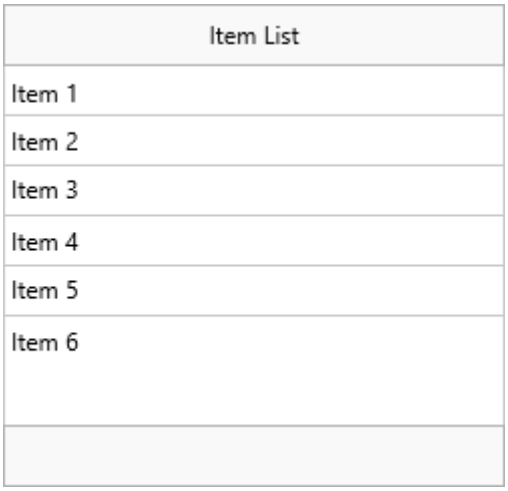
Adding items

When dropping an instance of TTMSFNCTableView on the form, you'll notice it already has a set of items with the name of cars. To get started with an empty view, remove all items from the collection. The tableview will automatically update itself accordingly.

Items can be added at designtime, or programmatically. To add items at designtime, open the items editor and click add. When an item is added, the tableview re-calculates and updates the lookupbar, sections and other related elements, depending on the item settings.

Programmatically adding items can be done using the following code:

```
TMSFNCTableView1.Items.Clear;
TMSFNCTableView1.AddItem('Item 1');
TMSFNCTableView1.AddItem('Item 2');
TMSFNCTableView1.AddItem('Item 3');
```



The tableview also exposes a set of methods to load items from a file or a stream.

```
TMSFNCTableView1.LoadFromStrings
TMSFNCTableView1.LoadFromFile
TMSFNCTableView1.LoadFromStream
```

All methods have a parameter that points to a list of items separated with a linebreak.

For each linebreak, a new item is inserted. To save a set of items, you can use the equivalent method.

TMSFNCTableView1.SaveToStrings
TMSFNCTableView1.SaveToFile
TMSFNCTableView1.SaveToStream

Item size

By default, each item automatically resizes itself to make sure there is enough space for the text and accessories. When adding HTML formatted text with linebreaks you'll notice the difference with other single-line text items. Below is a sample that demonstrates this on a default instance of TTMSFNCTableView.

TMSFNCTableView1.Items[1].Text := ''+TMSFNCTableView1.Items[1].Text+ ''+ '
Lovely car!';

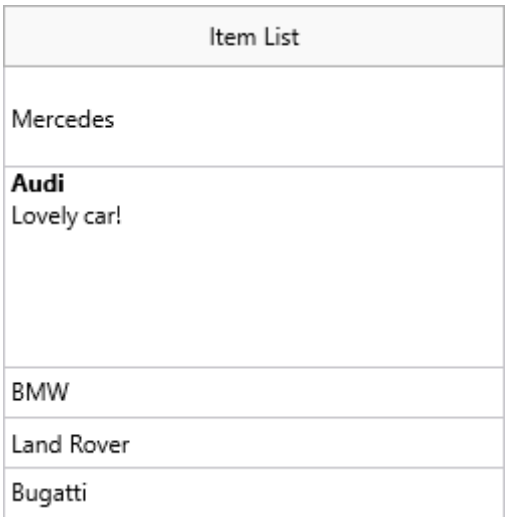
Item List
Mercedes
Audi Lovely car!
BMW
Land Rover
Bugatti

The behavior of fixed and dynamically resized items is controlled with the ItemAppearance.HeightMode propery. By default the HeightMode property is set to tvhmVariable. The other value of the HeightMode property is tvhmFixed which, in combination with the ItemAppearance.FixedHeight property, ensures the items are set to a specific Height. Below is a sample that shows setting the HeightMode to tvhmFixed. Notice that the wordwrapped text is no longer completely visible.

Item List
Mercedes
Audi
BMW
Land Rover
Bugatti

Alternatively you can set a height for each item separately. Each item has a height property which has a default value of -1. Any value other than -1 will force the item height to this value. Below is a sample that shows how to achieve this.

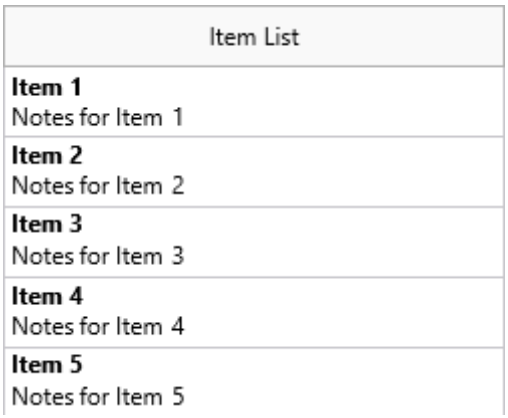
```
TMSFNCTableView1.Items[0].Height := 50;
TMSFNCTableView1.Items[1].Text := '<b>'+TMSFNCTableView1.Items[1].Text+ '</b>'+ '<br/>Lovely
car!';
TMSFNCTableView1.Items[1].Height := 100;
```



Item HTML template

Each item has a text property which can contain HTML formatted text. A sample of formatted text can be found in the previous chapter. Alternatively, a HTML template can be used to format each item text with placeholders. Each item has a HTMLTemplateItems collection that contain the values of the placeholder values defined in the ItemAppearance.HTMLTemplate string value. Below is a sample that demonstrates this.

```
var
  it: TTMSFNCTableViewItem;
  I: Integer;
begin
  TMSFNCTableView1.Items.Clear;
  TMSFNCTableView1.ItemAppearance.HTMLTemplate := '<b><#CAPTION></b><br/><#NOTES>';
  for I := 0 to 4 do
  begin
    it := TMSFNCTableView1.Items.Add;
    it.HTMLTemplateItems.Values['CAPTION'] := 'Item ' + inttostr(I + 1);
    it.HTMLTemplateItems.Values['NOTES'] := 'Notes for ' + it.HTMLTemplateItems.Values['CAPTION'];
  end;
end;
```



If you want to change the notes for item 4 you can then simply set the HTMLTemplateItems value property with the placeholder name defined in the ItemAppearance.HTMLTemplate property.

```
TMSFNCTableView1.Items[3].HTMLTemplateItems.Values['NOTES'] := 'Hello World!';
```

Item accessories

Each item has a set of accessories available. Below is a list of supported accessories.

- **Detail**: a detail accessory representing an image set with the ItemAppearance.AccessoryDetailBitmaps collection. The detail accessory typically indicates a detail control is assigned to the item.
- **Progress**: a progress accessory, based on a maximum of 100. The value of the progress bar can be set with the AccessoryProgress property at item level.
- **Button**: A clickable button, containing a text.
- **Badge**: A badge representing information to the user in a circular or rounded rectangular shape.
- **Custom**: An area that can be customized by the user using custom drawing.

The global accessory appearance can be customized under ItemAppearance. Accessory details, specific to an item, can be customized with the Accessory* properties at item level. Clickable accessories can be defined with the Interaction.AccessoryClickDetection property. By default the Button and Custom accessory type is enabled.

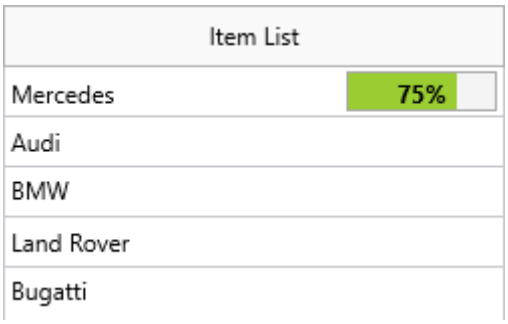
Below are some samples of setting an accessory.

Detail

```
TMSFNCTableView1.ItemAppearance.AccessoryDetailBitmaps.AddBitmapFromFile('MyImage.png');
TMSFNCTableView1.Items[0].Accessory := tviaDetail;
TMSFNCTableView1.Items[0].AccessoryWidth := 16;
TMSFNCTableView1.Items[0].AccessoryHeight := 16;
```

Progress

```
TMSFNCTableView1.Items[0].Accessory := tviaProgress;
TMSFNCTableView1.Items[0].AccessoryProgress := 75;
TMSFNCTableView1.Items[0].AccessoryFontColor := gcBlack;
```



Custom

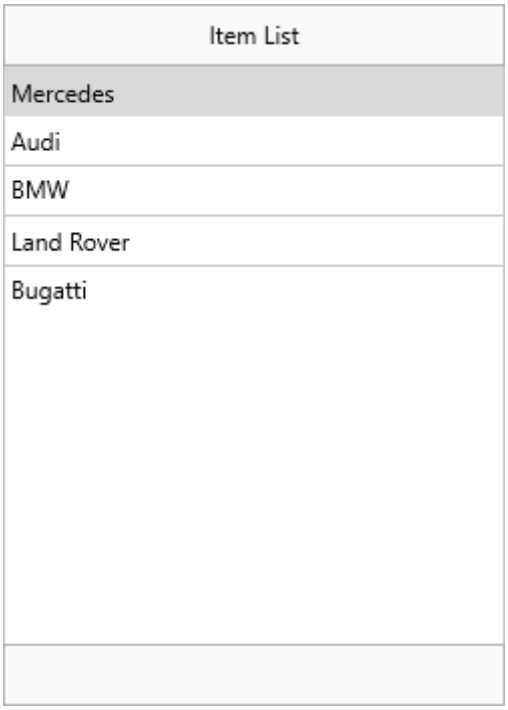
```
TMSFNCTableView1.Items[0].Accessory := tviaCustom;

procedure TForm1.TMSFNCTableView1DrawItemCustomAccessory(Sender: TObject;
  AGraphics: TTMSFNCGraphics; ARect: TRectF; AItem: TTMSFNCTableViewItem);
var
  r: TRectF;
begin
  r := ARect;
  InflateRectEx(r, -2, -2);
  AGraphics.Fill.Color := gcLightgoldenrodyellow;
  AGraphics.Stroke.Color := gcDarkred;
  AGraphics.DrawEllipse(r);
end;
```

Item detail

The item has a set of properties to control its appearance, as well as its behaviour. Each item has a DetailControl property that can be used to assign a control that is shown when clicking on the item. The tableview also has a DefaultItem property, which is basically the same as a normal item but acts only as a template. Each property that is set on the default item is inherited in newly added items. The DefaultItem property has a DetailControl property as well that can be used for each item.

Clicking on an item with a DetailControl property assigned will show the control with a slide-in effect. A back button will appear in the upper left corner of the header. Below is a sample that demonstrates this.

```
TMSFNCTableView1.Items[0].DetailControl := Panel1;
```



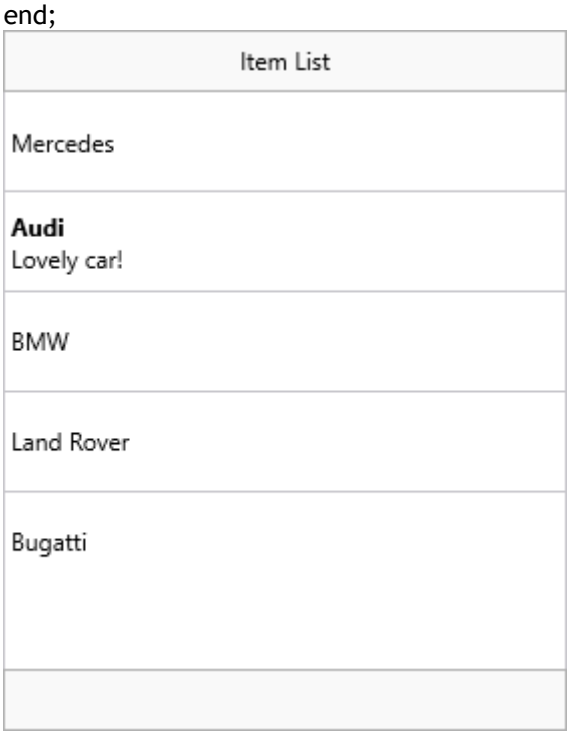
Clicking on the back button will then return to the main view.

Item more options

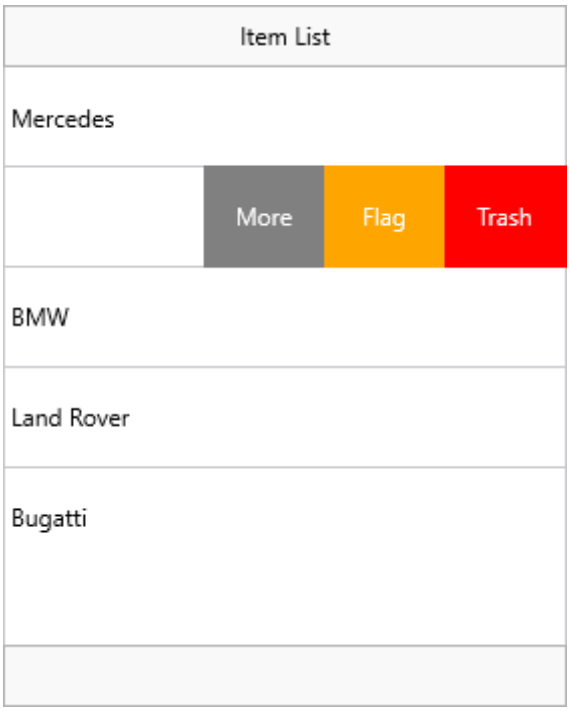
The tableview has a MoreOptions collection and MoreOptionAppearance. The MoreOptions collection is used to show additional buttons at the right side of an item after swiping it from right to left or vice versa. The MoreOptions collection is a global collection, so all buttons are identical for all items. The MoreOptionAppearance holds a fill, stroke and font for a global appearance, but each more option collection item can be further customized. Below is a sample that demonstrates how to show more options on a swiping action.

```
var
  mo: TTMSFNCTableViewMoreOption;
begin
  TMSFNCTableView1.BeginUpdate;
```

```
  TMSFNCTableView1.Items[1].Text := '<b>'+TMSFNCTableView1.Items[1].Text+ '</b>'+ '<br/>Lovely
car!';
  TMSFNCTableView1.Items[1].WordWrapping := False;

  mo := TMSFNCTableView1.MoreOptions.Add;
  mo.Text := 'More';
  mo.Color := gcGray;
  mo.BorderColor := gcGray;
  mo.FontColor := gcWhite;

  mo := TMSFNCTableView1.MoreOptions.Add;
  mo.Text := 'Flag';
  mo.Color := gcOrange;
  mo.BorderColor := gcOrange;
  mo.FontColor := gcWhite;

  mo := TMSFNCTableView1.MoreOptions.Add;
  mo.Text := 'Trash';
  mo.Color := gcRed;
  mo.BorderColor := gcRed;
  mo.FontColor := gcWhite;

  TMSFNCTableView1.ItemAppearance.HeightMode := tvhmFixed;
  TMSFNCTableView1.ItemAppearance.FixedHeight := 50;
  TMSFNCTableView1.EndUpdate;
end;
```



Now when swiping from right to left on any item the more option buttons will appear.



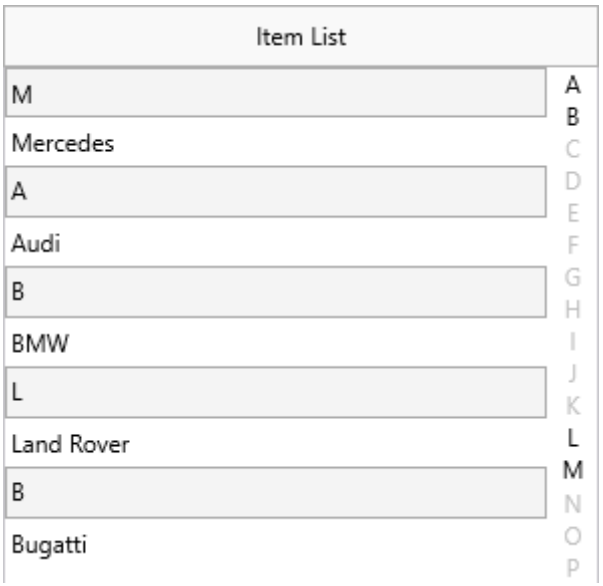
Clicking any of the more option buttons will trigger the OnItemMoreOptionClick event. Also note that the text is being shifted to the left, to ensure enough space is provided for buttons to be displayed.

Categories

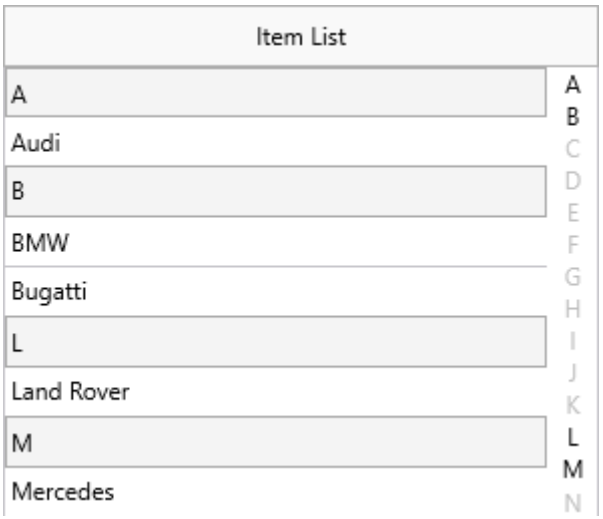
The tableview can divide items into different categories. The CategoryType property, which has a default value of tvctNone, is responsible for this. The CategoryType can be set to one of the following values:

- Alphabetic: Adds categories in alphabetic order.
- Alphanumeric: Adds categories in alphabetic order and includes numbers as well.
- Custom: Adds categories based on the Categories collection. Each item has a separate CategoryID property that can be linked to one of the categories in the Categories collection.

Below is a sample that shows the result after setting the CategoryType property to AlphaBetic on a default TTMSFNCTableView instance.



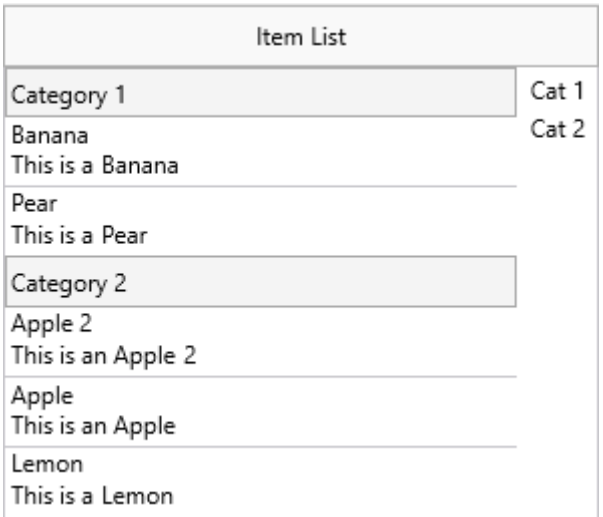
Note that there are new items (sections) added to the list that divide the items into separate categories. You will also notice that there are 2 of the “B” category which indicates the list itself is not sorted yet. To sort the items, call TMSFNCTableView1.Sort;



Sorting can be done in ascending or descending order with optional case sensitivity. To add custom categories, you can set the CategoryType to custom and add items to the Categories collection. Items that are added have a separate CategoryID property that is used in combination with the Categories collection to divide them. Below is a sample that demonstrates how to use custom categories.

```
var
  it: TTMSFNCTableViewItem;
  cat: TTMSFNCTableViewCategory;
begin
  TMSFNCTableView1.BeginUpdate;
  TMSFNCTableView1.Items.Clear;
  TMSFNCTableView1.LookupBar.Width := 40;
  TMSFNCTableView1.CategoryType := tvctCustom;
```

```
  cat := TMSFNCTableView1.Categories.Add;
  cat.Id := 0; // all items with CategoryID 0 belong to this category
  cat.Text := 'Category 1';
  cat.LookupText := 'Cat 1';

  cat := TMSFNCTableView1.Categories.Add;
  cat.Id := 1; // all items with CategoryID 0 belong to this category
  cat.Text := 'Category 2';
  cat.LookupText := 'Cat 2';

  it := TMSFNCTableView1.Items.Add;
  it.Text := 'Pear<br/>This is a Pear';
  it.CategoryID := 0;

  it := TMSFNCTableView1.Items.Add;
  it.Text := 'Banana<br/>This is a Banana';
  it.CategoryID := 0;

  it := TMSFNCTableView1.Items.Add;
  it.Text := 'Apple<br/>This is an Apple';
  it.CategoryID := 1;

  it := TMSFNCTableView1.Items.Add;
  it.Text := 'Lemon<br/>This is a Lemon';
  it.CategoryID := 1;

  it := TMSFNCTableView1.Items.Add;
  it.Text := 'Apple 2<br/>This is an Apple 2';
  it.CategoryID := 1;

  TMSFNCTableView1.Items.Sort;

  TMSFNCTableView1.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 text property of the item.

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 OnManualLookupCategory 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:

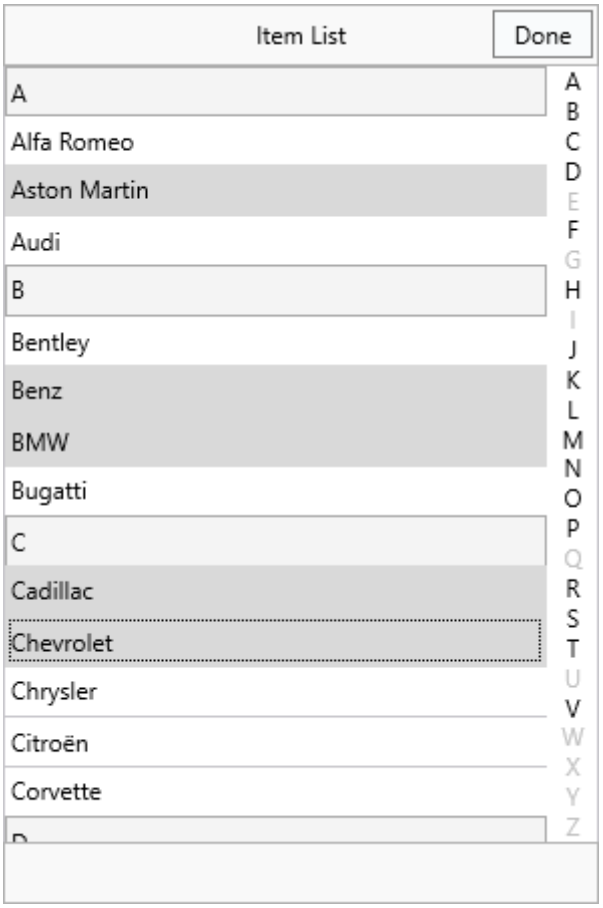
```
TMSFNCTableView1.LookupCategory(ACharacter);
//Performs a lookup of an AlphaBetic / AlphaNumeric category with a specific character.

TMSFNCTableView1.LookupCustomCategory(ACharacterID);
//Performs a lookup of a Custom category with a specific characterID.
```

Additionally, focusing the item and typing characters will also start a lookup, navigating to the item that matches the lookup string. Press escape or enter to reset the lookup search string.

Editing

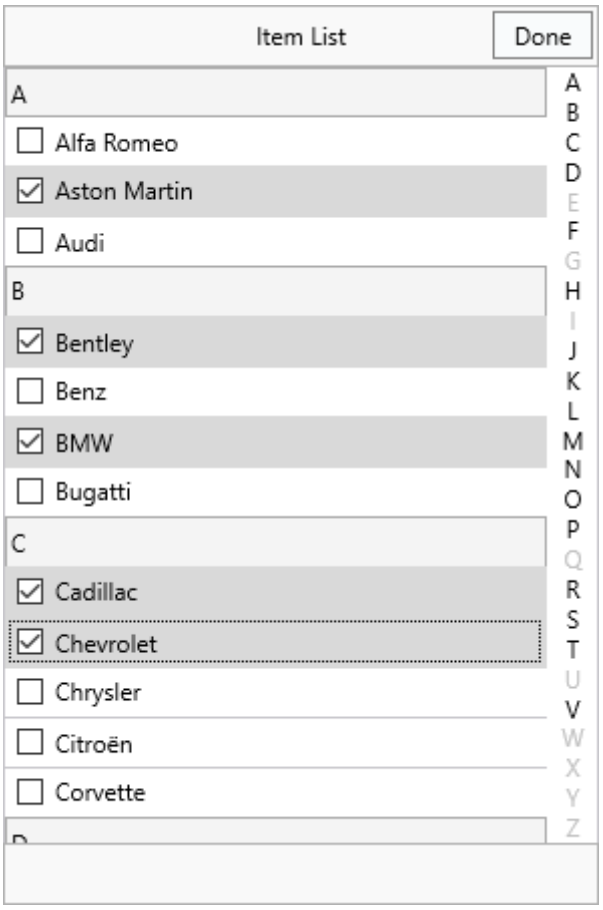
The tableview has a feature that enables checking multiple items at once. To enable this feature, set the Interaction.ShowEditButton to true. Clicking the button will enable edit mode, where

clicking the item selects it, along with other items that were already selected. Clicking an item again will unselect it. If editing mode is enabled, the items remain checked, but as soon as editing stops, the items are deselected.



Clicking the done button resets the multiple selection state of the items. To retrieve the items that are selected, use the SelectedItemCount function and the SelectedItems property. Clicking on the edit button will visually only change the button in the header. If you also want to visually indicate a checkmark / checkbox button you can set the DefaultItem.CheckType to tvictCheckBox.

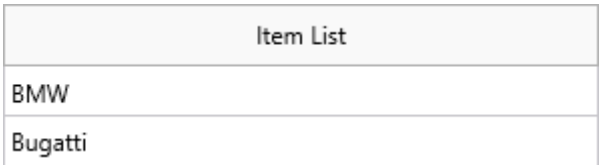
```
TMSFNCTableView1.CategoryType := tvctAlphaBetic;
TMSFNCTableView1.Interaction.ShowEditButton := True;
TMSFNCTableView1.DefaultItem.CheckType := tvictCheckBox;
TMSFNCTableView1.LoadFromFile('cars.txt');
```



Filtering

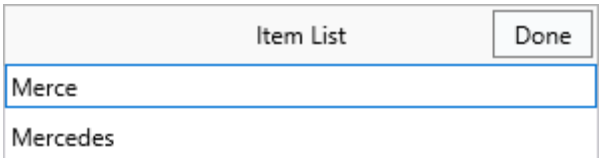
The tableview supports filtering on programmatic level as well as through an edit box. To filter, use the Filter property and add conditions that needs to be applied to the filter operation. Below is a sample that demonstrates how to filter programmatically.

```
var
  f: TTMSFNCTableViewFilterData;
begin
  f := TMSFNCTableView1.Filter.Add;
  f.Condition := 'B*';
  TMSFNCTableView1.ApplyFilter;
end;
```



Filtering can also be applied visually, by enabling the filter button under Interaction. With Interaction.ShowFilterButton, the filter button becomes visible. Clicking the button will show an edit box that allows typing. The filter operation that is applied via typing in the edit box is indentical to the one that is applied programmatically.

TMSFNCTableView1.Interaction.ShowFilterButton := True;

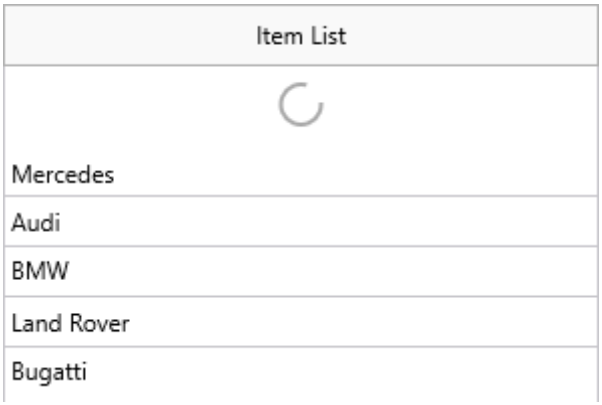


Click on the done button to stop filtering.

Reload

The tableview also supports a reload mechanism, which is visually represented as a circular progressbar (optionally marquee progress) that waits until the tableview reload process is finished. Below is a sample that demonstrates this.

1) Set TMSFNCTableView1.Reload.Enabled := True; which will enable the ability to reload the list.
2) Reload progress is started by dragging the list down, which calls the OnStartReload event, or by programmatically calling StartReload.
3) In the OnStartReload event you can start a thread which performs a task.
4) After the thread is finished, call TMSFNCTableView1.StopReload; which will then call the OnStopReload event.



Important methods, properties and events

Properties

BackButton	Public access to the backbutton, which is shown when navigating to the detail, when a detail control is assigned to one or multiple items and an item has been clicked.
BitmapContainer	A container of images that is used in combination with items and/or HTML formatted text.
Categories	A collection of categories that are used when the CategoryType property is set to tvctCustom and items are linked through the CategoryID property.
CategoryAppearance	The appearance of the categories that divide

	items into different groups.
CategoryType	The type of categories that divide items into different groups.
Checked[AItemIndex: Integer]	Gets or Sets a Boolean whether the item is checked or not based on the index of the item.
CheckedItem[AItem: TTMSFNCTableViewItem]	Gets or Sets a Boolean whether the item is checked or not based on a reference to the item.
DefaultItem	The default item, used as a template for newly added items.
DoneButton	Access to the done button, displayed when editing is started or when filtering is started.
EditButton	Access to the edit button, displayed when Interaction.ShowEditButton is set to true.
Filter	Programmatic access to filter operations.
FilterButton	Access to the filter button, displayed when Interaction.ShowFilterButton is set to true.
Footer	The footer of the tableview, which has optional HTML formatted text.
Header	The header of the tableview, which has optional HTML formatted text and acts as a container for the edit, filter, back and done buttons.
Interaction	The various interaction capabilities of the tableview.
ItemAppearance	The general appearance of an item.
ItemIndex	The current selected item.
Items	The collection of items.
LookupBar	The lookupbar settings of the tableview.
MoreOptionAppearance	The appearance of the more option buttons configured in the MoreOptions collection.
MoreOptions	The MoreOptions collection: a collection of buttons that are displayed inside the item after a swipe gesture from right to left over the item.
Reload	The reload settings of the tableview after performing a swipe gesture from top to bottom on the tableview.
SelectedItem	The current selected item.
SelectedItems[AIndex: Integer]	A specific selected item in case multiselection is enabled or editing is enabled.
VerticalScrollBarVisible	Enables or disables a vertical scrollbar. The scrollbar is only visible if the items exceed the total available content height of the tableview.

Methods

AddItem(AText: string = '')	Adds an item with an optional text parameter.
ApplyFilter	Applies the filter set in the Filter collection.
CheckAllItems	Checks all items.
ClearSelection	Clears the current item selection.
CopyToClipboard(ATextOnly: Boolean = False)	Copies the selected items to the clipboard, with an optional parameter to select text only.
CutToClipboard(ATextOnly: Boolean = False)	Copies the selected items to the clipboard and removes them afterwards.
DetailControlActive	Returns a Boolean to determine if the detail

	control is active or not.
DisableInteraction	Disables interaction with the tableview.
EnableInteraction	Enables interaction with the tableview.
GetCheckedItems	Gets an array of checked items.
GetItemsFromClipboard	Gets a collection of items that are stored in the clipboard.
GetSelectedItems	Gets an array of selected items.
HideDetailControl	Hides the detail control.
HideMoreOptions	Hides the more option buttons.
IsItemSelectable(AItem: TTMSFNCTableViewItem)	Returns a Boolean to determine if an item is selectable or not.
LoadFromFile(AFileName: string)	Loads the tableview from a file.
LoadFromStream(AStream: TStream)	Loads the tableview from a stream.
LoadFromStrings(AStrings: TStrings)	Loads the tableview from a string list.
LookupCategory(ACategory: String)	Looks up a specific category based on a character.
LookupCustomCategory(ACategoryID: Integer)	Looks up a specific category based on a custom category ID.
LookupItem(ALookupString: String; ACaseSensitive: Boolean = False; AAutoSelect: Boolean = False)	Looks up an item based on a specific lookup string, optionally case-sensitive. Has the ability to automatically select the item if the item is found.
PasteFromClipboard	Pastes items from the clipboard to the tableview.
ReloadProgress	The progress that indicates the reload time when the reload progress mode is tvrpmManual.
RemoveFilter	Clears the filter from the tableview..
RemoveFilters	Clears the filter from the tableview, and additionally removes all filter data from the filter collection.
RemoveItem(AItem: TTMSFNCTableViewItem)	Removes a specific item from the tableview.
SaveToFile(AFileName: string; ATextOnly: Boolean = True)	Saves the tableview to a file.
SaveToStream(AStream: TStream; ATextOnly: Boolean = True)	Saves the tableview to a stream.
SaveToStrings(AStrings: TStrings)	Saves the tableview to a string list.
ScrollToItem(AItemIndex: Integer)	Scrolls the tableview to a specific item based on an item index.
SelectedItemCount	Returns the selected item count.
SelectItem(AItemIndex: Integer)	Selects a specific item based on an item index.
SelectItems(AItemIndexes: TTMSFNCTableViewIntegerArray)	Selects an array of item indexes.
ShowDetailControl(AItemIndex: Integer = -1)	Shows the detail control assigned to a specific item. When the AItemIndex = -1 the DefaultItem detail control is used, when it is assigned.
ShowMoreOptions(AItem: TTMSFNCTableViewItem)	Shows more options on a specific item.
Sort(ACaseSensitive: Boolean = True; ASortingMode: TTMSFNCTableViewItemsSortMode = tvismAscending)	Sorts the items, with optional case sensitivity and ascending / descending sort mode.
StartEditMode	Starts edit mode in the tableview.
StartFiltering	Starts filter mode in the tableview.
StartReload	Starts a reload operation in the tableview.

StopEditMode	Stops edit mode in the tableview.
StopFiltering	Stops filter mode in the tableview.
StopReload	Stops the active reload operation.
ToggleEditMode	Toggles the tableview edit mode or non-edit mode.
UnCheckAllItems	Unchecks all items.
UpdateReloadProgress(AProgress: Single; AAutoStopReload: Boolean = True)	Updates the reload progress when the reload progress mode is set to tvrpmManual.
XYToAccessoryItem(X, Y: Single)	Returns an accessory item under a specific X and Y coordinate.
XYToItem(X, Y: Single)	Returns an item under a specific X and Y coordinate.
XYToItemIndex(X, Y: Single)	Returns an item index under a specific X and Y coordinate.
XYToMoreOption(X, Y: Single)	Returns a more option button under a specific X and Y coordinate.

Events

OnAfterApplyFilter	Event called after a filter is applied.
OnAfterDrawItem	Event called after an item is drawn.
OnAfterDrawItemCheck	Event called after an item checkbox is drawn.
OnAfterDrawItemIcon	Event called after an item icon is drawn.
OnAfterDrawItemText	Event called after an item text is drawn.
OnBeforeApplyFilter	Event called before a filter is applied.
OnBeforeDrawItem	Event called before an item is drawn.
OnBeforeDrawItemCheck	Event called before an item checkbox is drawn.
OnBeforeDrawItemIcon	Event called before an item icon is drawn.
OnBeforeDrawItemText	Event called before an item text is drawn.
OnBeforeItemHideDetailControl	Event called before the detail control of an item is hidden.
OnBeforeItemShowDetailControl	Event called before the detail control of an item is shown.
OnCategoryClick	Event called when a category is clicked.
OnCategoryCompare	Event called when the category collection is sorted and categories are begin compared.
OnDrawItemCustomAccessory	Event called when a custom accessory needs to be drawn.
OnItemAccessoryClick	Event called when an item accessory is clicked.
OnItemAnchorClick	Event called when an item anchor (HTML formatted text) is clicked.
OnItemCheckChanged	Event called when the item checkbox state changes.
OnItemClick	Event called when an item is clicked.
OnItemCompare	Event called when the item collection is sorted and items are being compared.
OnItemDblClick	Event called when an item is double-clicked.
OnItemHideDetailControl	Event called when an item detail control is hidden.
OnItemMoreOptionClick	Event called when an item more option button is clicked.
OnItemSelected	Event called when an item is selected.
OnItemShowDetailControl	Event called when an item detail control is shown.

OnManualLookupCategory	Event called when a lookup category is clicked in the lookupbar, and the AutoLookup property is set to False. Through this event you can manually lookup a specific category.
OnStartReload	Event called when a reload operation is started.
OnStopReload	Event called when a reload operation is stopped.
OnVScroll	Event called when a vertical scroll operation occurs.