TMS TAdvResponsiveList, TDBAdvResponsiveList DEVELOPERS GUIDE

June 2020

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

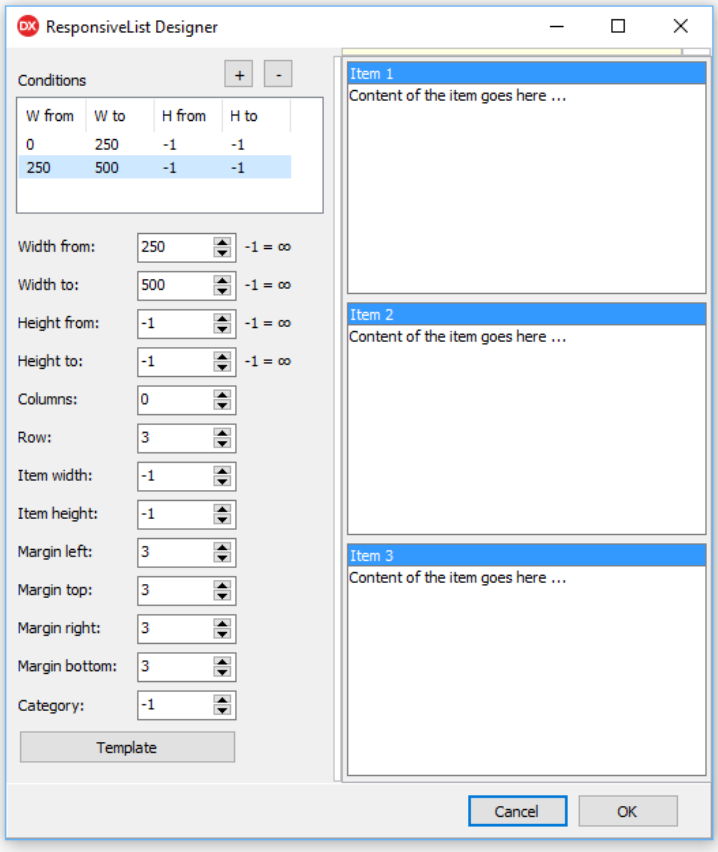
TAdvResponsiveList

Introduction

The component TAdvResponsiveList brings responsive design (https://en.wikipedia.org/wiki/Responsive_web_design) methodology to native Windows applications. While responsive design's origin is in accommodating a web page layout dynamically to the size of the browser, similar techniques can also prove useful in native Windows application design. As such application are typically offered in a resizable window, it can greatly improve the user experience when the layout adapts to the size the user chooses for the application.

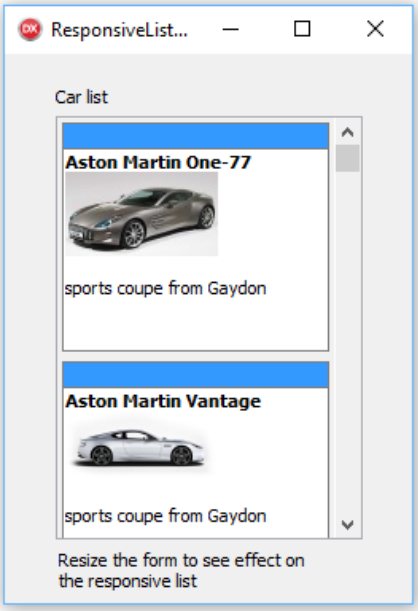
Architecture

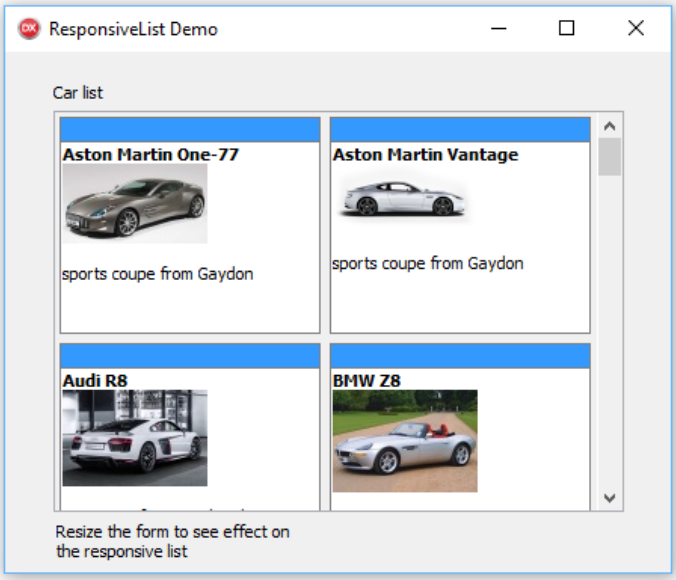
TAdvResponsiveList is designed around a configurable matrix of cells depending on the client area of the control. This configurability is controlled by a collection of conditions. For each condition, the range of client width and client height can be set for which a given number of columns or rows is used or a cell width or height is used.
This conditions collection can be programmatically configured but can of course also be edited at design-time with this design-time editor:

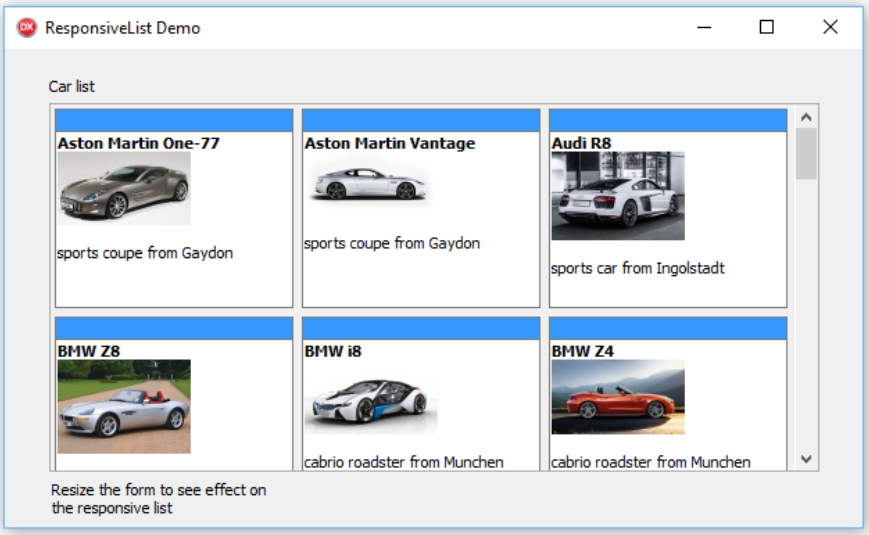


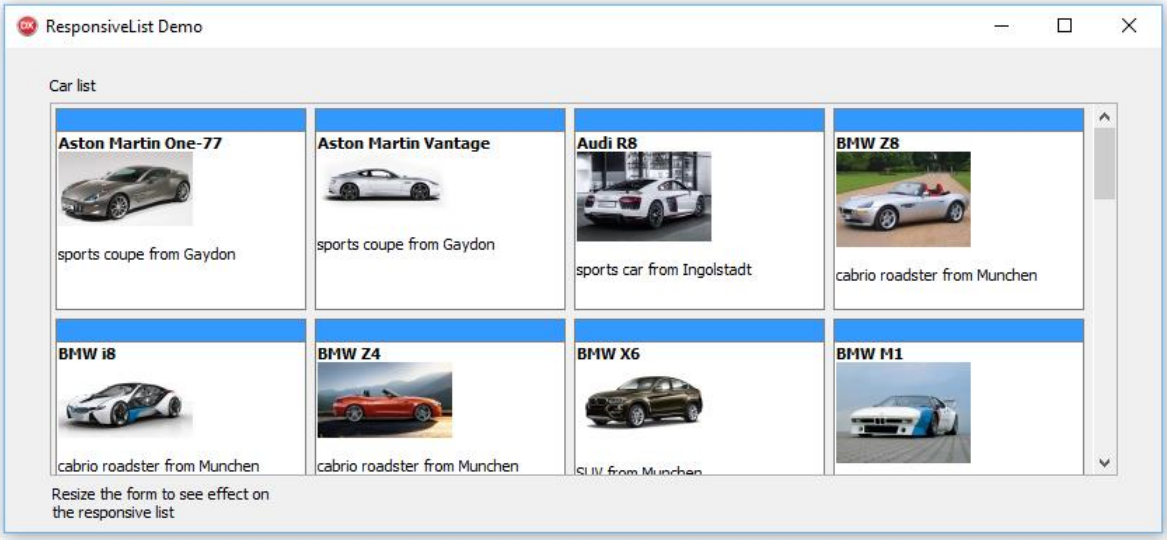
In the top left listview, the conditions are listed. In this example, 4 conditions for 4 different width ranges are configured, i.e. from 0..250pixels, from 250..500pixels, from 500..750pixels and from 750 to any higher width. In this sample, in the smallest width range, the number of columns is set to 1, the next width it is set to 2, then 3 and finally 4 columns when the width is larger than 750pixels. The item height is then in all circumstances configured to 150pixels. As the item width is set to -1, this means the width of items will size proportionally with the width of the control. This width could have been set to a fixed width in pixels as well.

To understand the basics of the architecture better, this leads to following behaviour:









Conditions

Programmatically, these conditions can be setup via the collection: TAdvResponsiveList.Conditions

The TResponsiveCondition item in this collection has following properties:

property Category: integer : identifier of the category to which a condition belongs

property Columns: integer : number of columns to use when this condition is selected

property FooterTemplate: string : template for the item footer

property HeaderTemplate: string : template for the item header

property HeightFrom: integer : control height min. value required for this condition to be selected (when value is -1, this means accept any control height)

property HeightTo: integer : control height max. value required for this condition to be selected (when value is -1, this means accept any control height)

property ItemWidth: integer : item width to use when this condition is selected. When value is -1, the entire control width is used.

property ItemHeight: integer : item height to use when this condition is selected. When value is -1, the entire control height is used.

property Margins: TMargins : margin to use between items for this condition

property Rows: Integer : number of rows to use when this condition is selected

property Template: string : template for the item content

property WidthFrom: integer : control width min. value required for this condition to be selected (when value is -1, this means accept any control width)

property WidthTo: integer : control width max. value required for this condition to be selected (when value is -1, this means accept any control width)

property Tag: NativeInt : integer identifier for the condition

Items

The TAdvResponsiveList has a collection of items that are rendered in the list. The item can be fully custom drawn, but it already supports rendering of HTML formatted text and with this also the rendering of images and hyperlinks. In addition to this, an item can have a header and a footer. The properties of the base item class are:

BorderColor: TColor
Color of the border of the item

BorderStyle: TBorderStyle
Sets the border style to bsNone or bsSingle

Color: TColor
Sets the background color of the item

Content: string
Sets the text content (can be HTML formatted content) of the item

FooterColor: TColor
Sets the color of the footer. When the color is clNone, no footer is drawn

FooterTextColor: TColor
Sets the font color of the footer

FooterText: string
Sets the text of the footer

HeaderColor: TColor
Sets the color of the header. When the color is clNone, no header is drawn

HeaderTextColor: TColor
Sets the font color of the header

HeaderText: string
Sets the text of the header

Height: integer
Defines the height of the item when the height type is different from isAuto

HeightType: TItemSizeType
Can be:
-isAuto: height of the item is automatically determined by the conditions
-isFixed: height of the item is fixed in pixels
-isPerc: height of the item is fixed in percentage
-isFill: height of the item is equal to the control height

SelectedBorderColor: TColor
Sets the border color of the item when it is in selected state

SelectedColor: TColor
Sets the background color of the item when it is in selected state

SelectedTextColor: TColor
Sets the text color of the item when it is in selected state

Shadow: Boolean
When true, the item is drawn with a shadow

Tag: NativeInt
Generic item tag property

TextColor: TColor
Color of the item text in normal state

Visible: Boolean
Controls the item visibility state

Width: Integer
Defines the width of the item when the width type is different from isAuto

WidthType: TItemSizeType
Can be:
-isAuto: width of the item is automatically determined by the conditions

-isFixed: width of the item is fixed in pixels
-isPerc: width of the item is fixed in percentage
-isFill: width of the item is equal to the control height

Responsive templated items

Not only the size of the item can be responsively determined but also the formatting of the header, footer and content. This is done via responsive templates. Part of the conditions is a template for item content, header and footer. A template is a HTML formatted string with value placeholders. The control will then automatically resolve the value placeholders with the actual values held by an item.

This way, in a small version of an item, it could show less text, i.e. in the condition for the width of the control being <= 250pixels, the condition template could have been set to:

<b>(#TITLE)<b>
(#PRICE)

while for a control width > 250pixels, the template could have been set to:

<b>(#TITLE)<b>
(#PRICE)
(#DESCRIPTION)

When the item contains values for TITLE, PRICE and DESCRIPTION, the control will automatically resolve the correct content template according to the selected condition depending on the width of the control.

The values of the item are a NAME/VALUE pair collection where the VALUE is a variant type. For this particular case, the item's NAME/VALUE pairs could have been set via

```
var
  it: TResponsiveListItem;

it := AdvResponsiveList.Items.Add;

it.Values['TITLE'] := 'Lion King';
it.Values['PRICE'] := 123.456;
it.Values['DESCRIPTION'] := 'The Lion King is a 1994 American animated epic musical film produced by Walt Disney Feature Animation and released by Walt Disney Pictures.';

it := AdvResponsiveList.Items.Add;

it.Values['TITLE'] := 'Nemo';
it.Values['PRICE'] := 210.987;
it.Values['DESCRIPTION'] := 'Finding Nemo is a 2003 American computer-animated comedy-drama adventure film produced by Pixar Animation Studios';
```

The 3 templates that can be set via conditions are as such:

TResponsiveCondition.Template: string; = template for the item content

TResponsiveCondition.HeaderTemplate: string; = template for the item header (when used)

TResponsiveCondition.FooterTemplate: string; = template for the item footer (when used)

Filtering items in the TAdvResponsiveList

With the help of filtering, it is possible to only view items that meet certain filter criteria. These filter criteria are setup via AdvResponsiveList.FilterCondition with following properties:

FilterCondition.CaseSensitive: Boolean : when true, a case sensitive match is needed to withhold an item

FilterCondition.FilterData: TResponsiveListItemFilterDataSet : determines to use the filter condition for item header, footer, content

FilterCondition.FilterType: TResponsiveListItemFilterType = (mText, mEntireWord, mStartWord, mEndWord) : specifies to filter on any part of the text, entire words in the text, beginning of the text or end of the text

FilterCondition.Text: string : holds the filter criteria

When the filter criteria are set, filtering is started by calling:

AdvResponsiveList.UpdateFilter;

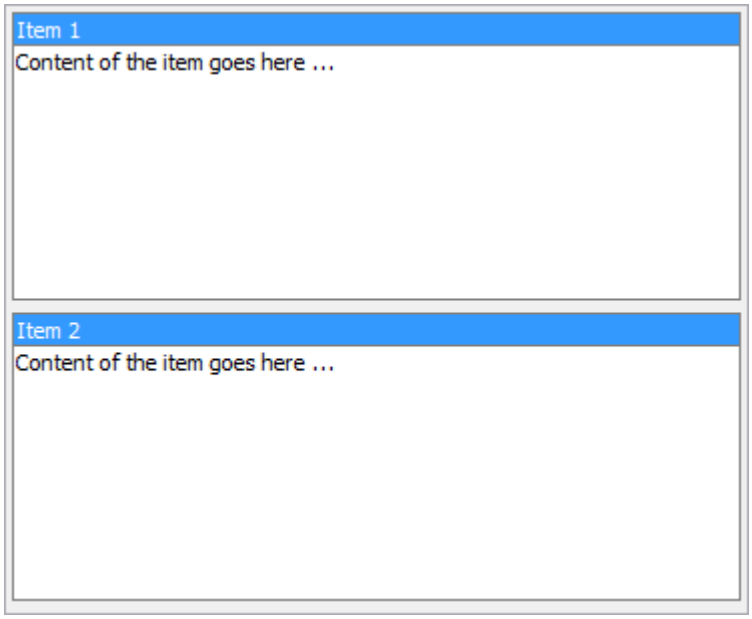
When a filter is active, it is removed again by calling

AdvResposiveList.ClearFilter;

Responsive lists in TAdvResponsiveList

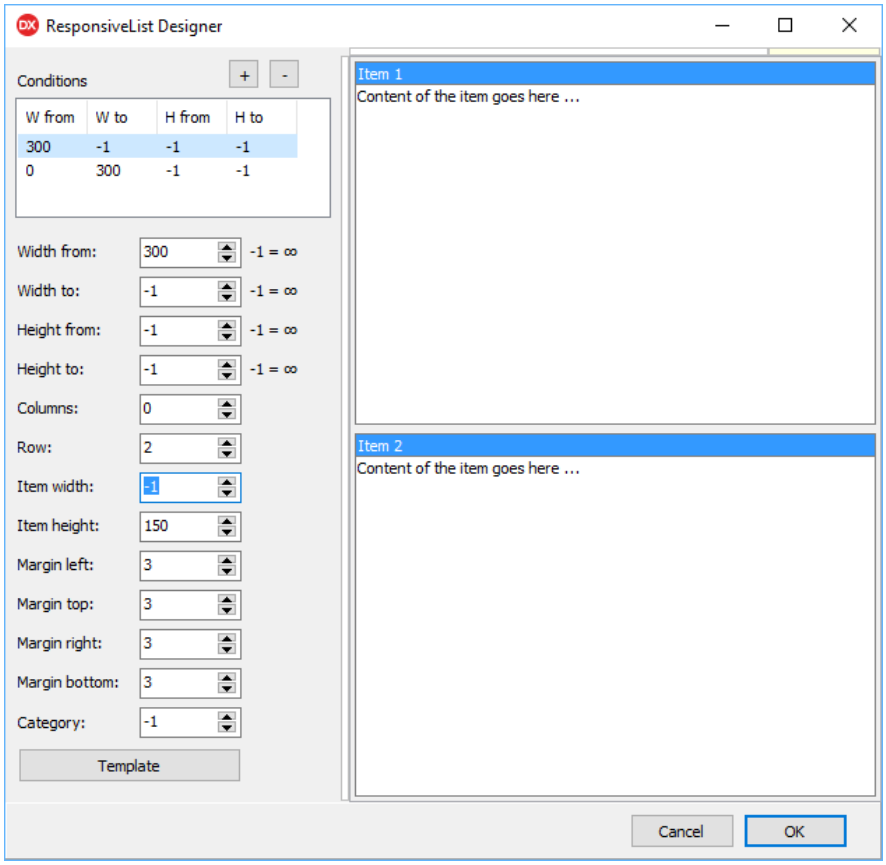
The number of possibilities of using TAdvResponsiveList becomes sheer unlimited when hosting a TAdvResponsiveList within a TAdvResponsiveList.

To illustrate this concept, let’s start with a TAdvResponsiveList with 2 items:

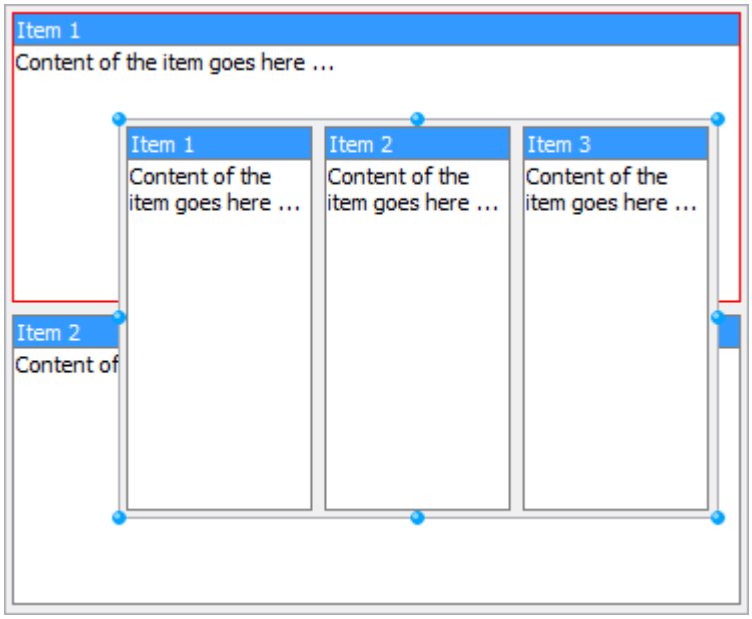


And configure 2 conditions:

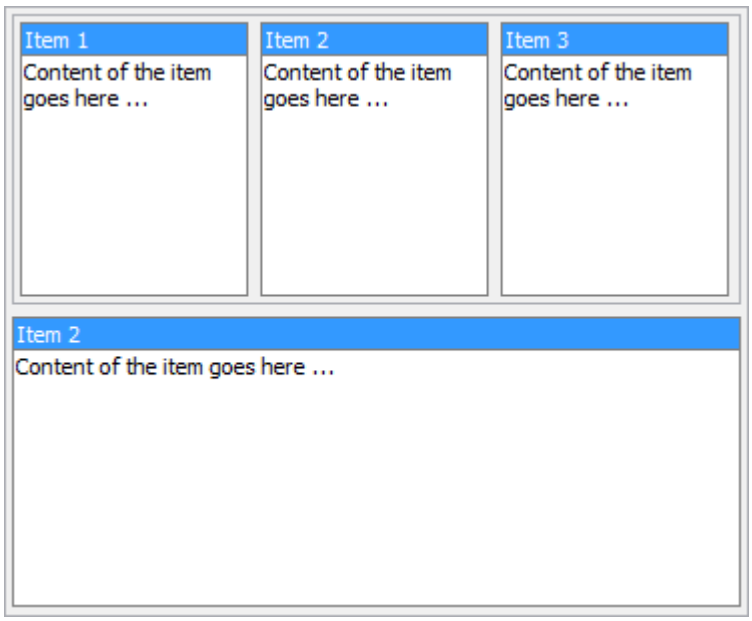
The first condition will render the list as a list of items in 2 rows when the width of the control is 300pixels or wider and a list of items in 2 columns when the width is smaller than 300.



Next drop on this TAdvResponsiveList a second TAdvResponsiveList:

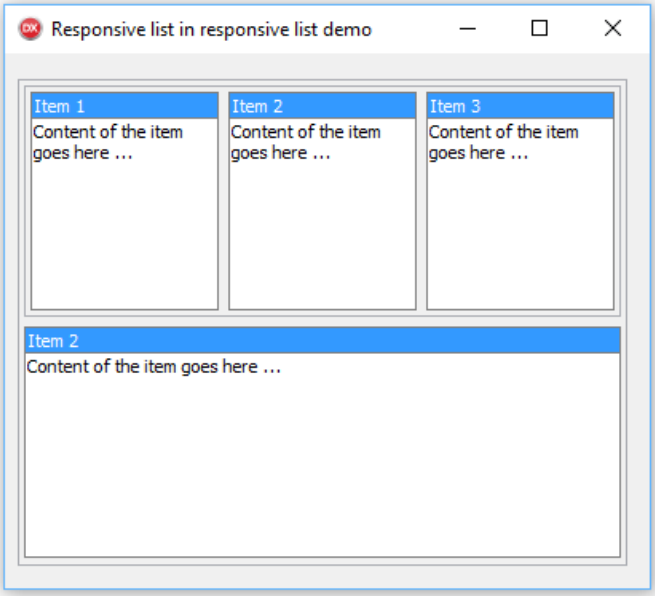


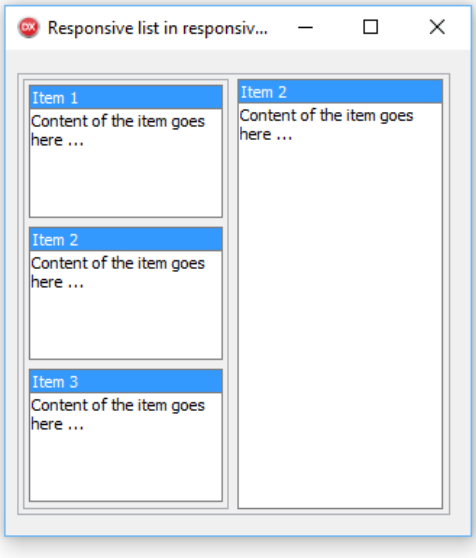
At design time, the item in the parent TAdvResponiveList where the 2nd TAdvResponsiveList is dropped on will display with a red border. At design time the 2nd TAdvResponsiveList can be dragged over the 2nd item. Once the parent TAdvResponsiveList item to connect the 2nd TAdvResponsiveList is chosen, you can set the child TAdvResponsiveList.Align to alClient and it will align to the chosen parent list item. When the application is run, this results in:



Now, on this child TAdvResponsiveList, we can again add conditions for responsive behaviour. In this case, we add the condition that for a width of 300pixels or higher, it will render its items in columns and for a width of less than 300pixels, it will render the items in rows.

With these conditions in place, resizing the parent TAdvResponsiveList will result in the child TAdvResponsiveList to adapt to its parent item and as such, also responsively adapt to render its items in rows:





Creating custom TAdvResponsiveList controls

It is straightforward to creating custom TAdvResponsiveList controls that use custom items.

To do so, create a custom TResponsiveListItem class that descends from TResponsiveListItem:

```
type
  TResponsiveListItemEx = class(TResponsiveListItem)
  private
    FPicture: TPicture;
    FCustomProp: string;
    procedure SetPicture(const Value: TPicture);
  protected
```

```
    procedure DrawItem(ACanvas: TCanvas; ATemplate: string; ARect: TRect); override;
    procedure PictureChanged(Sender: TObject);
  public
    constructor Create(Collection: TCollection); override;
    destructor Destroy; override;
    procedure Assign(Source: TPersistent); override
  published
    property Picture: TPicture read FPicture write SetPicture;
    property CustomProp: string read FCustomProp write FCustomProp;
  end;
```

and then with overriding the GetItemClass protected method in a descending control of TAdvResponsiveList, start using this custom item class:

```
  TAdvResponsiveListEx = class(TAdvResponsiveList)
  private
  protected
    function GetItemClass: TCollectionItemClass; override;
  published
  end;
```

with:

```
function TAdvResponsiveListEx.GetItemClass: TCollectionItemClass;
begin
  Result := TResponsiveListItemEx;
end;
```

In this example, we have added a TPicture property to the custom TResponsiveListItemEx class as well as a CustomProp string property. By then overriding the item's DrawItem() protected method, the item becomes responsible to draw itself within the responsive list:

```
procedure TResponsiveListItemEx.DrawItem(ACanvas: TCanvas; ATemplate: string;
  ARect: TRect);
begin
  inherited DrawItem(ACanvas, ATemplate, ARect);

  if Assigned(FPicture.Graphic) and not FPicture.Graphic.Empty then
    ACanvas.Draw(ARect.Left + 10, ARect.Top + 10, FPicture.Graphic);

  ACanvas.TextOut(ARect.Left, ARect.Top, CustomProp);
end;
```

As the default item class DrawItem() method is still called here, this means the custom item will draw a picture and text on top of the existing item.

TDBAdvResponsiveList

Introduction

The component TDBAdvResponsiveList adds the capability to automatically present data from a dataset in the responsive list UI.

The dataset is connected via the assignment of a datasource via TDBAdvResponsiveList.DataSource and the data from the items presented in the list is defined by the header, content and footer template.

Setting up TDBAdvResponsiveList

Assume there is a dataset with following fields: NAME, PRENAME, STREET, ZIPCODE, CITY connected via a datasource to the TDBAdvResponsiveList. To display the information in items, the condition's content template value could be set to:

TResponsiveCondition.Template := '<B>(#PRENAME) (#NAME)</B>
(#STREET)
(#ZIPCODE) (#CITY)'

When the dataset is active, the content template will be rendered using the dataset record values for PRENAME, NAME, STREET, ZIPCODE, CITY.

Note that it is also possible to display memo fields or blob fields holding a picture. For memo fields, nothing special needs to be done. When the memofield's name is MEMOFIELD, using the identifier (#MEMOFIELD) in the template is sufficient to let it render the content of the memo field.

To display the value of a blob field as picture in the item, the <IMG> tag can be used in the template to invoke the display as picture and the identifier for the image is the fieldname of the blob field. Suppose that the blob field is called PICTURE, the template could contain:

<IMG src="(#PICTURE)">

Example:

For a dataset of movie actors & actresses that has a field NAME, PRENAME, YOB (year of birth), PICTURE and MOVIE, we display the actor/actress name in the header and the year of birth and picture with movie title in the content of the item.

Therefore the templates become:

Header template:

(#PRENAME) (#NAME)

Content template:

Born: (#YOB)
<IMG src="(#PICTURE)">
Movie:
(#MOVIE)

The result becomes: