IW

TTIWResponsiveList - TTIWDBResponsiveList
DEVELOPERS GUIDE

April 2017

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

Table of contents

TTIWResponsiveList / TTIWDBResponsiveList availability

TTIWResponsiveList and TTIWDBResponsiveList are available as VCL components for Delphi and C++Builder.

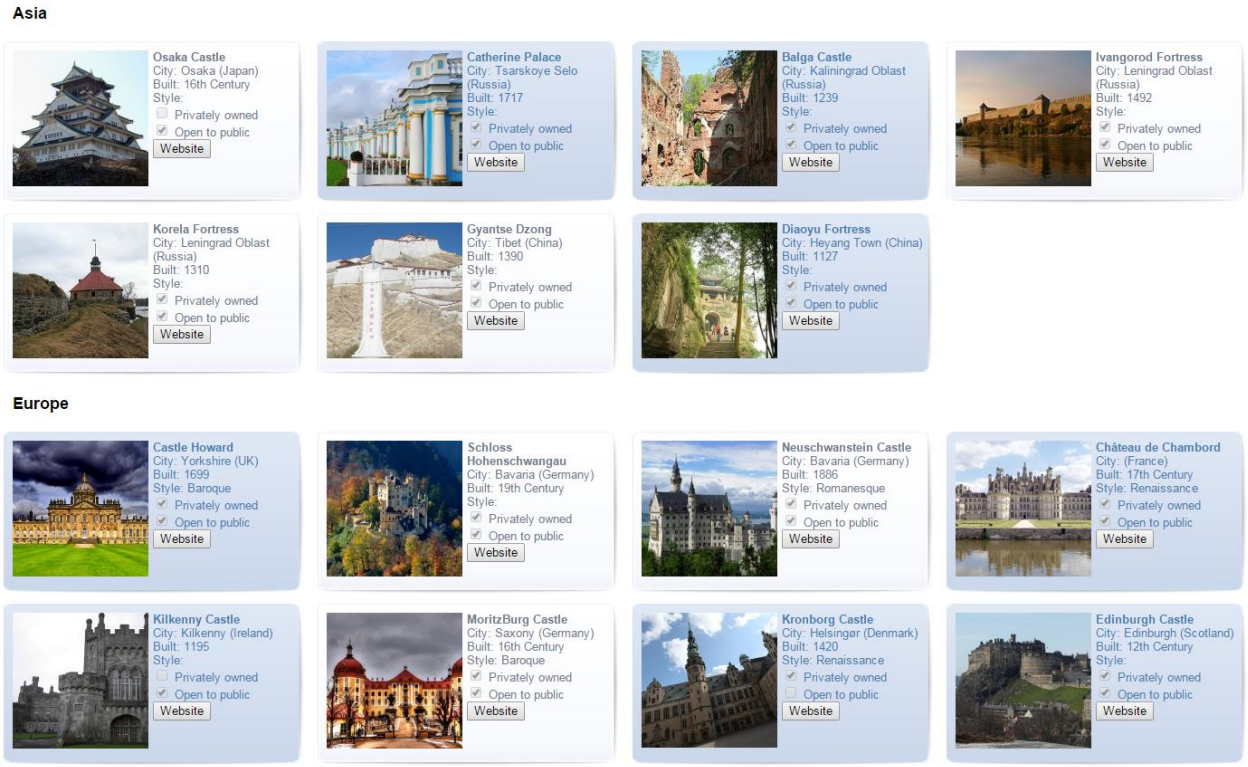
TTIWResponsiveList and TTIWDBResponsiveList are available for:
- Delphi 2009,2010,XE,XE2,XE3,XE4,XE5,XE6,XE7,XE8,10 Seattle,10.1 Berlin, 10.2 Tokyo
- C++Builder 2009,2010,XE,XE2,XE3,XE4,XE5,XE6,XE7,XE8,10 Seattle,10.1 Berlin, 10.2 Tokyo

IntraWeb 12.1.x, 14.0.x is required

TTIWResponsiveList and TTIWDBResponsiveList have been designed for and tested with: Windows Vista, 7, 8, 10 on IntraWeb 12.1.x or higher

Current version of TTIWResponsiveList, TTIWDBResponsiveList has been designed for and tested with IE 11, FireFox 35, Chrome 40, Safari 5, iOS 6 and Android mobile browsers. Some features may not be available in older browser versions.

TTIWResponsiveList / TTIWDBResponsiveList use



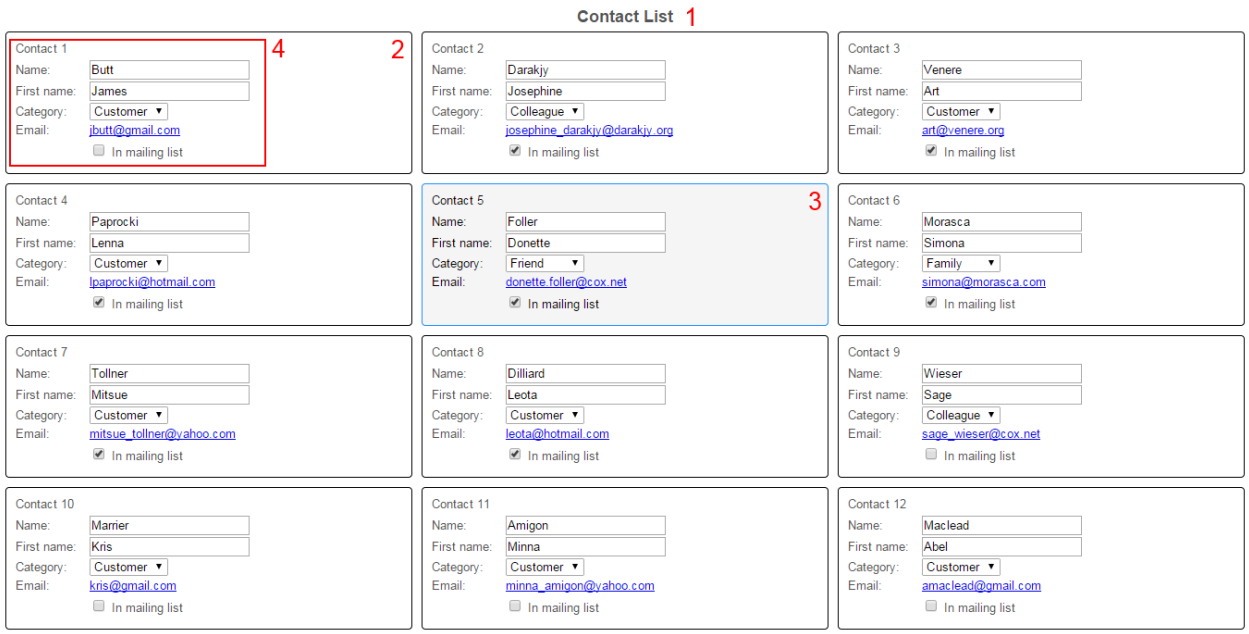
The TMS TTIWResponsiveList and TTIWDBResponsiveList components are designed to be used in all kinds of list type data presentation and editing in a browser. Data presented in the list can be database driven in the TTIWDBResponsiveList component and directly web application driven in TTIWResponsiveList. It is from the web application running on the server that this list presentation layer is generated along with JavaScript code that is executed in the browser on the client side. TTIWResponsiveList and TTIWDBResponsiveList have built-in support for asynchronous rendering, responsive layout and continuous scrolling. Finally, TTIWResponsiveList and TTIWDBResponsiveList share the underlying presentation layer and browser JavaScript generating code, making the end-user experience identical for both data-aware and non data-aware list.

Asynchronous rendering
The control supports full asynchronous rendering, inserting and updating of list items.

Responsive Layout
The items from the Items collection are displayed automatically in 1 or more columns depending on the available browser window width. By default the minimum number of columns is 1 (for mobile phone screens) and the maximum is 4 (for large desktop screens).

Continuous Scrolling
When the control is rendered for the first time only a part of the available items is rendered to fill the available browser window height and create a vertical scrollbar.
When scrolling to the end of the page, the remaining items are automatically added asynchronously, filling one extra screen at a time.

TTIWResponsiveList / TTIWDBResponsiveList organisation



1. List Title
The title text appears on top of the list.

2. Default Item
The default appearance of an item. The default appearance of an item can be customised by configuring the DefaultItem properties. The appearance of a single item can be customised by using ItemStyles.

3. Selected Item
One or more items can be selected. Selected items can be configured to have a custom appearance.

4. Item Content
The content of an item is fully customizable and supports data binding (TTIWDBResponsiveList only) as well as HTML, JavaScript and CSS.

TTIWResponsiveList / TTIWDBResponsiveList properties

BGColor: TIWColor; The background color of the control.
BGImage: string; The background image of the control. Can contain an absolute or relative URL to an image file. The BGColor property is ignored if this property is not empty.
BorderColor: TIWColor; The color of the control's border.
ClientScript: string; The content of this property is rendered as JavaScript in the browser. This can be useful if custom JavaScript is required for the Items[].Text (TTIWResponsiveList only) or DataTemplate (TTIWDBResponsiveList only).
DefaultItem: TIWItemStyle; The default style definition for an item.
ItemHeight: integer; The default height of an item.
ItemIndex: integer; The selected item (if MultiSelect is false, else the selected state is configured per item).
ItemSize: TIWItemSize; When set to isAuto the ItemHeight and ItemWidth values will act as a minimum size and the item will automatically be stretched to fit it's content. When set to isFixed the ItemHeight and ItemWidth values will act as fixed size and the item's content will be clipped to fit the item.
ItemStyles: TIWItemStyles; A collection of custom item styles.
ItemWidth: integer; The default width of an item. When set to 0 the item will automatically shrink or stretch based on the available width.
Layout: TIWListLayout; The layout of the items. llResponsive is default and recommend. The llFlow option is available for use with browsers that do not support responsive layout.
MultiSelect: Boolean; When set to true, multiple items can be selected at once. When set to false only a single item can be selected.
Padding: TIWPadding; The padding between items.
ResponsiveLayout: TIWResponsiveLayoutItems; Collection of responsive layout definitions. Sets the number of columns that are displayed for a pre-defined screen size. By
SectionHeight: integer; The default height of a section item.
Title: string; The text that appears on top of the list.

TIWItemStyle:
BackgroundImage: string; The background image of the control. Can contain an absolute or relative URL to an image file. The Color and ColorTo properties are ignored if this property is not empty.
BorderColor: TIWColor; The border color of the item.
BorderWidth: integer; The width of the item border. Set to 0 to remove the border.
Color: TIWColor; The color of the item.
ColorTo: TIWColor; The gradient color of the item.
Font: TIWFont; The font settings of the item.
RoundedBorderRadius: integer; The border rounding radius of the item. Set to 0 to disable rounding.
SelectBackgroundImage: string; The background image of the control in selected state. Can contain an absolute or relative URL to an image file. The Color and ColorTo properties are ignored if this property is not empty.
SelectBorderColor: TIWColor; The border color of the item in selected state.
SelectColor: TIWColor; The color of the item in selected state.
SelectColorTo: TIWColor; The gradient color of the item in selected state.
SelectFontColor: TIWColor; The font color of the item in selected state.

TIWResponsiveLayoutItem:
Columns: integer; The number of columns that are displayed in the list when the width of the browser window is equal or larger than the value defined in the MinScreenWidth property.
MinScreenWidth: integer; The minimum required browser window width to display the number of columns in the list as defined in the Columns property.

TTIWResponsiveList specific properties:

Items: TIWListItems; The collection of items that are displayed in the list.

TIWListItem:
Css: string; The css class that is used for the default appearance of the item. ItemStyle settings are discarded when a value is defined.
CssSelected: string; The css class that is used for the appearance of the item in selected state. ItemStyle settings are discarded when a value is defined.
IsSection: Boolean; When set to true the item is displayed as a section; a divider between items. The item starts on a new line and spans all columns of the list. The item that follows a section item also starts on a new line.
ItemStyle: string; The name of the ItemStyle item from the ItemStyles collection that is used for the appearance of the item.
Selected: Boolean; When set to true and MultiSelect is set to true the item is displayed in selected state.
Text: TStringList; The content of the item. Can contain HTML, JavaScript and CSS code.

TTIWDBResponsiveList specific properties:

DataSource: TDataSource; The TDataSource associated with the control.
DataTemplate: TStringList; The template definition that is used to display the list items. The template definition supports data binding and can contain HTML, JavaScript and CSS code. See the "TTIWDBResponsiveList templates" topic for detailed information on using templates.
ShowMemoFields: Boolean; When set to true the full text of a database memo field is displayed in the item's content, otherwise only "(MEMO)" is displayed.

TTIWResponsiveList / TTIWDBResponsiveList methods

AsyncItemAdd(AItem: TIWListItem);
Asynchronously add a new item to the list.

AsyncItemUpdate(AItem: TIWListItem);
AsyncItemUpdate(AItemIndex: integer);
Asynchronously update an existing item in the list.

AsyncUpdate();
Asynchronously update all items in the list.

AsyncSetAttribute(ItemIndex: integer; ControlID, AttributeName, AtrributeValue: string);
Asynchronously set/update an attribute value of an HTML control defined in the control's content/template.

AsyncGetAttributes(ItemIndex: integer; ControldIDs, AttributeNames: TStringList);
Asynchronously request one or more attributes from one or more HTML controls defined in the control's content/template. The results are returned through the OnAsyncGetAttributes event.

AsyncRemovAttribute(ItemIndex: integer; ControlID, AttributeName: string);
Asynchronously remove an attribute from an HTML control defined in the control's content/template.

AsyncExecuteJS(FunctioName: string; Parameters: TStringList);

Asynchronously execute an existing JavaScript function with the parameters from the Parameters list. Custom JavaScript functions can be added through the ClientScript property. JavaScript function results can be returned server-side with a call to "#HTMLNAME#ExecuteAsync(ItemIndex, ItemValue);" JavaScript function which will trigger the OnAsyncExectuteJS event.

By default there is a JavaScript function "#HTMLNAME#ExecuteAsync(ItemIndex, ObjectID);" available that returns an HTML object based on the ItemIndex and ObjectID parameters. The ItemIndex is the index of the item in the Items collection (TTIWResponsiveList only) or the index of the record in the DataSet (TTIWDBResponsiveList only). The ObjectID refers to the id attribute value of the HTML object defined in the item's content.

Sample: (Get the selected index of an HTML combobox defined in the Items[].Text)

```
//Add the HTML combobox to the item text

Items[0].Text.Add('<select id="OPTIONLIST"><option>1</option><option
selected>2</option><option>3</option></select>');


//Add the required custom JavaScript

TIWResponsiveList1.ClientScript :=
'function GetSelected(ItemIndex) {'
  + '  o = ' + TIWResponsiveList1.HTMLName + 'GetControl(ItemIndex,
"OPTIONLIST");'
  + '  if (o)'
  + '    ' + TIWResponsiveList1.HTMLName + 'ExecuteAsync(ItemIndex,
o.selectedIndex);'
  + '}';


//Call AsyncExecuteJS from an OnAsyncClick event of a TIWButton

procedure TIWForm1.IWButton1AsyncClick(Sender: TObject;
  EventParams: TStringList);
var
  Params: TStringList;
begin
  Params := TStringList.Create;
  Params.Add('0');
  TIWResponsiveList1.AsyncExecuteJS('GetSelected', Params);
  Params.Free;
end;


//Update a label's text with the value returned by the OnAsyncExecuteJS
event

procedure TIWForm1.TIWResponsiveList1AsyncExecuteJSEvent(Sender: TObject;
  EventParams: TStringList; ItemIndex: Integer; ReturnValue: string);
begin
  IWLabel1.Text := ReturnValue;
end;
```

TTIWDBResponsiveList specific method:

UpdateRecord;
Update the currently selected item with the new values from data-bound HTML input controls as defined in the DataTemplate. This event requires that the DataSet associated with the control is in edit mode. The updated values are retrieved asynchronously and the OnAsyncUpdated event is triggered if the operation was successful. See the "TTIWDBResponsiveList templates" topic for detailed information on using templates.

TTIWResponsiveList / TTIWDBResponsiveList events

OnItemClick(Sender: TObject; Index: Integer);
Event triggered when an item was clicked.

OnItemRender(Sender: TObject; Index: Integer; var ItemStyle: string);
Event triggered just before each item is rendered. The ItemStyle parameter can be used to change the style of the item.

OnAsyncControlEvent(Sender: TObject; EventParams: TStringList; EventType, ControlID: string; ItemIndex: Integer);
Asynchronous event triggered for HTML controls in the Items[].Text (TTIWResponsiveList only) or DataTemplate (TTIWDBResponsiveList only) that have the "#HTMLNAME#ControlEventHandler(EventObject, EventType);" JavaScript function assigned to an event. See the "TTIWDBResponsiveList templates" topic for a sample.

OnAsyncExecuteJS(Sender: TObject; EventParams: TStringList; ItemIndex: Integer; ReturnValue: string);
Asynchronous event that can be triggered from custom JavaScript code through calling "#HTMLNAME#ExecuteAsync(ItemIndex, ItemValue);". Custom JavaScript code can be added through the ClientScript property.

OnAsyncGetAttributes(Sender: TObject; EventParams: TStringList; ItemIndex: Integer; ControlIDs, AttributeNames, AttributeValues: TStringList);
Asynchronous event triggered after calling the AsyncGetAttributes method. The ControlIDs parameter contains the list of control id's that the attributes where requested for. The AttributeNames parameter contains the list of attribute names and the AttributeValues parameter contains the values of the requested attributes.

OnAsyncItemClick(Sender: TObject; EventParams: TStringList; Index: Integer);
Asynchronous event triggered when an item was clicked.

OnAsyncScroll(Sender: TObject; EventParams: TStringList);
Asynchronous event triggered if extra items are rendered after the list in the browser is scrolled.

TTIWDBResponsiveList specific events:

OnAsyncUpdated(Sender: TObject; ItemIndex: integer);
Asynchronous event triggered after calling the UpdateRecord method. This event should be used to perform a post on the DataSet associated with the control.

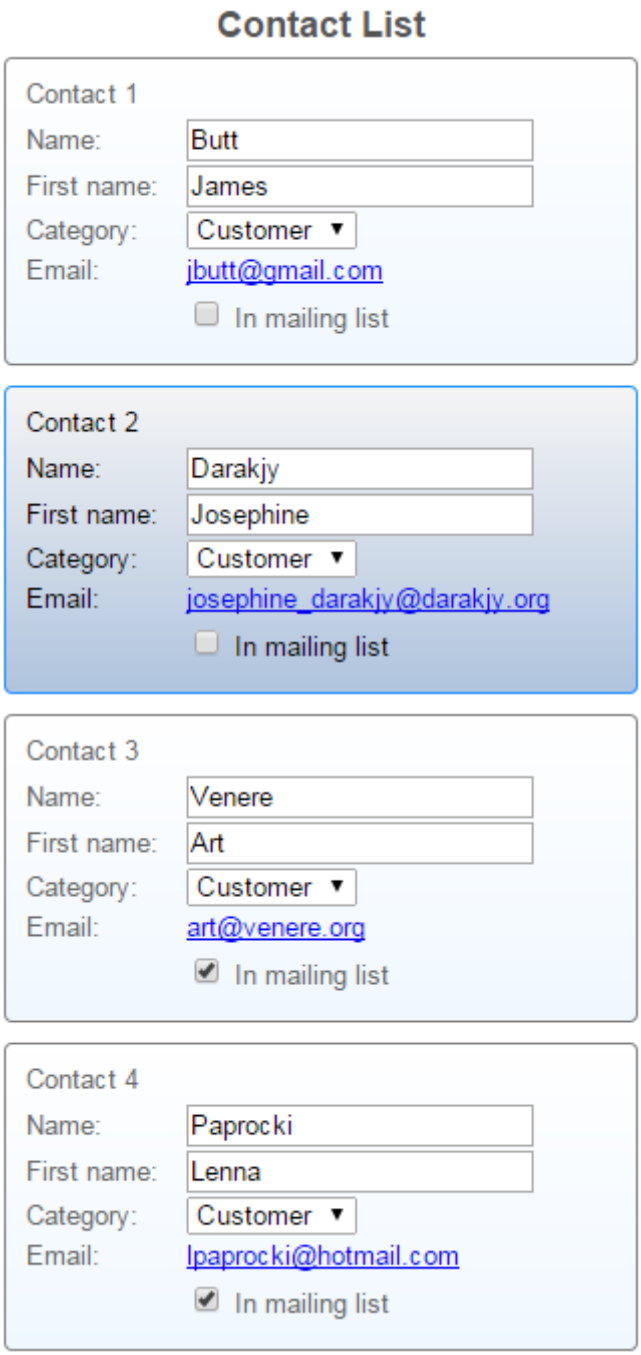
OnGetDataTemplate(Sender: TObject; ItemIndex: Integer; FieldName: string; var Data: string; var DisplayType: TIWDisplayType);
Event triggered for each data field placeholder in the DataTemplate. The Data parameter can be used to change the field value data. The DisplayType parameter can be used to set how the field

data is displayed in combination with the DataTemplate. See the “TTIWDBResponsiveList templates” topic for detailed information on using templates.

DisplayType options are:
dtCheckBox: For Boolean fields in combination with an HTML checkbox only: returns the string “ checked” if the Data is true and an empty string otherwise.
dtImage: For Blob type fields that contain image data only: returns the path to a temporary image file that contains the data from the blob field.
dtList: The field should be rendered as a combo box. The OnGetListData event is triggered and can be used to add the items that should be displayed in the list.
dtText: The default value, the field data is unchanged and rendered as text

OnGetListData(Sender: TObject; FieldName: string; var ListItems: TStringList);
Event triggered when DisplayType was set to dtList for the corresponding FieldName value. The ListItems parameter can be used to provide the items that are displayed in the list. The item for which the value is equal to the field data will automatically be selected. The item text and value can be provided by using name-value pairs. This requires that open and closing “SELECT” HTML tags are present in the DataTemplate. See the “TTIWDBResponsiveList templates” topic for detailed information on using templates.

TTIWDBResponsiveList templates



Templates can be used to display the dataset records in a uniform way. HTML, JavaScript and CSS can be included.
Data binding is performed by using placeholders (i.e.: "<#FIELDNAME>") in the DataTemplate. Placeholders will automatically be replaced with the value of that field for the respective item. By default the field value is displayed as text string. For different field types the DisplayType can be set via the OnGetDataTemplate event that is triggered for each placeholder. If DisplayType is set to dTList, an additional event, OnGetLisData, is triggered via which the list items can be provided.

To enable updating of records set the id attribute of the HTML input controls identical to the field name. Updating can be performed via the UpdateRecord method. The updated values are retrieved and the OnAsyncUpdated event is triggered if successful.

JavaScript events can be handled by a call to the "#HTMLName#ControlEventHandler(EventObject, EventType);" JavaScript function. This function will then trigger the OnAsyncControlEvent event. The EventObject parameter should contain the JavaScript event object. The EventType parameter can be used to differentiate between different JavaScript events.

Sample:

Database fields:
BLOBFIELD: a blob field that contains image data
TEXTFIELD: a standard text or integer field
OPTIONFIELD: a field that can only contain a limited number of values
BOOLEANFIELD: a field that contains a boolean value

```
//Add the template definition to the DataTemplate:

procedure TIWForm2.IWAppFormCreate(Sender: TObject);
begin
  TIWDBResponsiveList1.DataTemplate.Add('<img src="<#BLOBFIELD>">');
  TIWDBResponsiveList1.DataTemplate.Add('Text: <b><#TEXTFIELD></b>');
  TIWDBResponsiveList1.DataTemplate.Add('<br>Input: <input id="TEXTFIELD"
type="text" value="<#TEXTFIELD>">');
  TIWDBResponsiveList1.DataTemplate.Add('<br>Select: <select
id="OPTIONFIELD"><#OPTIONFIELD></select>');
  TIWDBResponsiveList1.DataTemplate.Add('<br><input type="checkbox"
id="BOOLEANFIELD" <#BOOLEANFIELD>> Checkbox');
  TIWDBResponsiveList1.DataTemplate.Add('<br>Button: <input id="button"
type="button" value="Click" onclick="' + TIWDBResponsiveList1.HTMLName +
'ControlEventHandler(event, ''Click'')">');
end;


//Use the OnGetDataTemplate to select the required DisplayType per field

procedure TIWForm1.TIWDBResponsiveList1GetDataTemplate(Sender: TObject;
  ItemIndex: Integer; Fieldname: string; var Data: string;
  var DisplayType: TIWDisplayType);
begin
  if FieldName = 'BLOBFIELD' then
    DisplayType := dtImage;

  if FieldName = 'BOOLEANFIELD' then
    DisplayType := dtCheckBox;

  if FieldName = 'OPTIONFIELD' then
    DisplayType := dtList;
end;


//Provide the list items for the OPTIONFIELD via the OnGetListData event.

procedure TIWForm1.TIWDBResponsiveList1GetListDataEvent(Sender: TObject;
  FieldName: string; var ListItems: TStringList);
```

```
begin
  if FieldName = 'OPTIONFIELD' then
  begin
    // Display value and field value separated by =
    ListItems.Add('Four=4');
    ListItems.Add('Six=6');
    ListItems.Add('Eight=8');
    ListItems.Add('Ten=10');
  end;
end;
```

Sample:

```
//Call UpdateRecord from an OnAsyncClick event of TIWButton

procedure TIWForm1.IWButton1AsyncClick(Sender: TObject;
  EventParams: TStringList);
  TIWDBResponsiveList1.DataSource.DataSet.Edit;
  TIWDBResponsiveList1.UpdateRecord;
end;


//Post the updates to the DataSet in the OnAsyncUpdated event

procedure TIWForm1.TIWDBResponsiveList1AsyncUpdated(Sender: TObject;
  ItemIndex: Integer);
begin
  TIWDBResponsiveList1.DataSource.DataSet.Post;
end;



//Use the OnAsyncControlEvent event to catch JavaScript events from HTML
objects defined in the DataTemplate

procedure TIWForm1.TIWResponsiveList1AsyncControlEvent(Sender: TObject;
  EventParams: TStringList; EventType, ControlID: string; ItemIndex:
Integer);
begin
  IWLabel1.Text := EventType + ' event on ' + ControlID + ' for item ' +
IntToStr(ItemIndex);
end;
```