Date:25-may-2023

SEGMENT FEATURE APP

version: 1.0.1

OVERVIEW

The Segment Feature App highlights the usage and capabilities of the Segment widget in the Quantum stack. The app highlights simple actions like swipe to delete and pull to refresh. It also highlights complex actions such as row animations during scroll, expand and collapse of a row on tap, and coupling two segments to scroll together.

**Requirements:**

* Volt MX Iris
* [Volt MX Foundry](https://manage.hclvoltmx.com/)

**Devices:**

* Mobile

**Platforms:**

* Android
* IOS

**Features:**

* Row Animation
* Expand/Collapse of a row item
* Swipe to Delete/Mark.
* Pull to Refresh.
* Coupled Segments.

**GETTING STARTED**

**Prerequisites**

Before you start previewing the Segment Feature app on your Mobile device, ensure that you have the following:

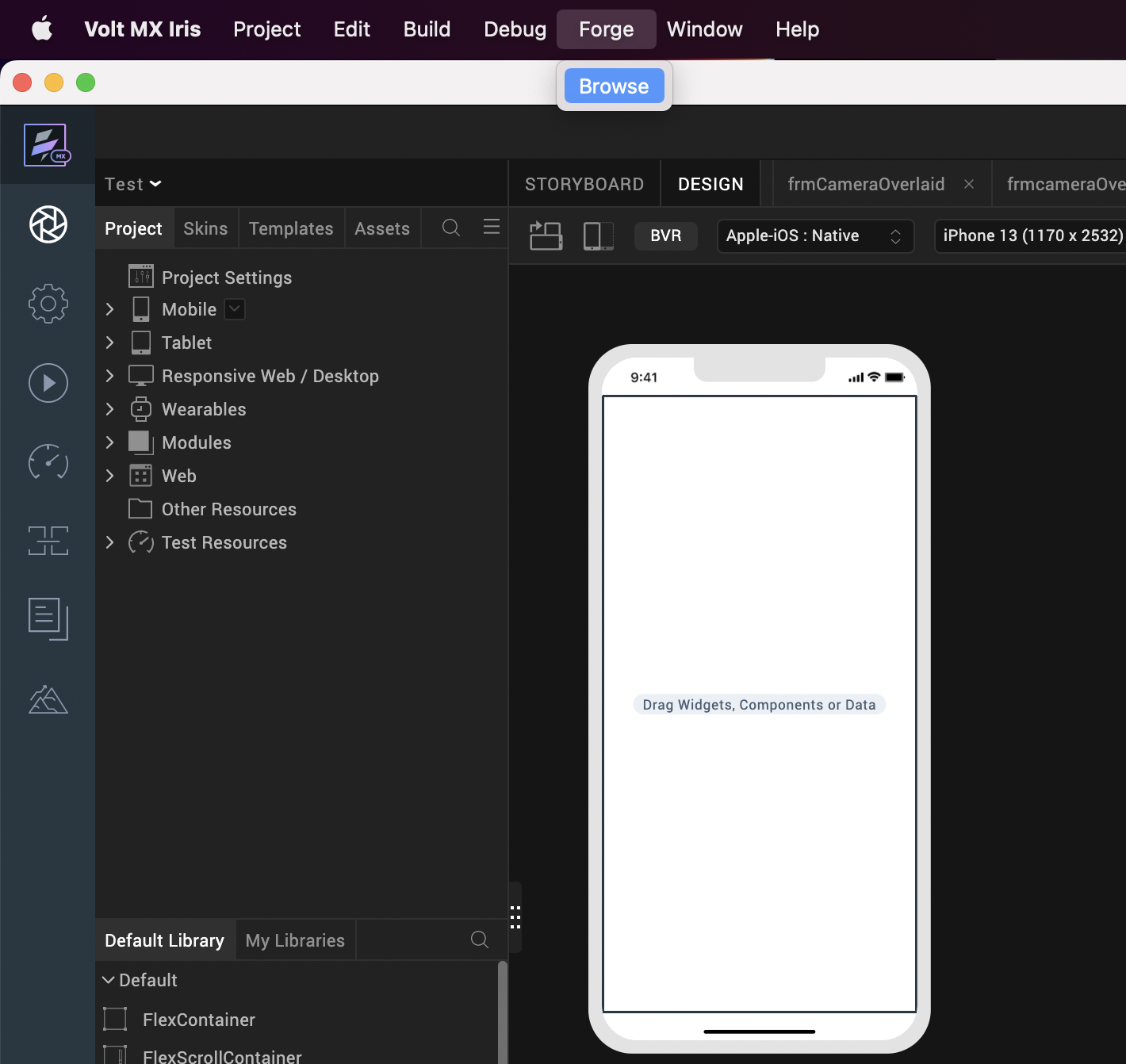
1. A Voltmx Account

2. Voltmx Iris.

**Import the App**

To import the Segment Feature app into your workspace, follow these steps:

1. Open Voltmx Iris..
2. On the main menu bar select Forge → Browse. The Voltmx Forge window appears.



1. Search for the Segment Feature app, and then click Import to Workspace. The app is imported to your workspace.  
     
   A dialog box is shown once the app is imported. Click OK.
2. Switch to your project containing the Segment Feature App. To switch to your project, click File → Open → Reference Architecture → <project\_name>

**Building and previewing the app**

After performing all the above steps, you can build your app and run it on your device. For more information, you can refer to the [Building and Viewing an Application](https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Iris/iris_user_guide/Content/Cloud_Build_in_VoltMX_Iris.html#cloud) section of the Volt MX User Guide.

You can then run your app to see the Segment Feature App Functionality

**App Functionality**

The landing page of the Segment Feature app is the Home screen. The Home screen displays the following :

A screenshot of a phone

Description automatically generated with medium confidence

Based on the button you select, the app navigates you to the respective app feature screen.

* Row Animation
* Expand/Collapse
* Swipe to Delete
* Pull to Refresh
* Coupled Segments

## Row Animation

In Segment rows, you can **move** (translate), **rotate**, or **scale** the child widgets. For example, you can display a rotating logo, or a hand icon that simulates waving by moving back and forth.

This screen of the app makes use of the **scale** animation provided by the segment widget. As you scroll down in the message list layout of the segment widget, the rows appearing from the bottom will scale from size zero and the center of the row, to the actual size and position of the row.

A screenshot of a computer

Description automatically generated with low confidence

### **Expand-Collapse**

This page shows a product list with some default product meta-data. This page shows a product list with some default metadata. When you tap on a row, it expands to show a brief product description. When you tap on an already expanded row, it collapses to hide the product description.

Note: This view does not follow an accordion menu functionality. Hence, you can expand a row even when another row is already in an expanded state.

A screenshot of a phone

Description automatically generated with low confidence

### **Swipe to Delete/Mark**

A screenshot of a phone

Description automatically generated with medium confidence

This page contains a list of messages with swipe functionality. If you swipe right to left on a message row, it slides 20% to the left to reveal a **Delete** button underneath. A tap on the **Delete** button removes the entry from the segment records.

A screenshot of a phone

Description automatically generated with medium confidence

If you swipe left to right on a message row, it slides 20% to the right to reveal a **Mark as Unread** button underneath. A tap on the **Mark as Unread** button dismisses the animated presentation and restores the segment to its previous state. In the code, the call back method for this action is kept blank for implementing the desired functionality.

A screenshot of a phone

Description automatically generated with medium confidence

### **Pull to Refresh**

This page shows a product list with some default product metadata. Initiating a pull from top to bottom on the segment and releasing the same triggers an action.

On triggering of this action, an alert is shown to the user stating "Refresh Action Triggered".

In the code, the call back method for the Pull to Refresh Functionality is blank for implementing any desired functionality based on the app they're developing.

A screenshot of a phone

Description automatically generated with medium confidence

**Coupled Segments**

This page shows two product lists next to each other for comparison. Both lists contain some default metadata for the products. There is also a **switch** at the top of the screen that **locks**/**unlocks** the two segments.

* When the switch is on, swiping from top to bottom or bottom to top scrolls both the segments together.

A picture containing text, font, number, screenshot

Description automatically generated

* When the switch is off, swiping from top to bottom or bottom to top scrolls the segment on which the swipe is being performed.

A picture containing text, font, number, screenshot

Description automatically generated

## Configuring the UI of the Segment Feature App

The app comes with a pre-configured UI. You don't need to make any changes to the code to make the app work. However, if you want to modify the UI of the app, you can follow the guidelines below.

The UI of the app is based on three templates:

* SMS Message
* Product List
* Product List Mini

### **SMS Message**

The **Row Animation** and **Swipe to Delete/Mark** screens use a segment to show a UI that you can create or modify under the **Templates** tab.

You can follow these guidelines to modify the **Template** for the **SMS Message UI**:

1. Import the app into Visualizer.
2. Go to the **Templates** tab in the **Project Explorer**.
3. In the provided list, expand **Mobile** > **Segments** > **tempSmsMessage**
4. **tempSmsMessage** is the template used in the **SMS Message UI**.
5. Expand **tempSmsMessage** to explore the template.

Click on the **BVR** button to see all the widgets that are beyond the device width.

A screenshot of a computer

Description automatically generated

* As shown in the image, an outer container, **flxTempRowWrapper**, is used to carry the elements inside the template.
* **flxTempRowWrapper** is divided into three parts-
  + - * flxMarkRead
      * flxRowWrapper
      * flxDelRow

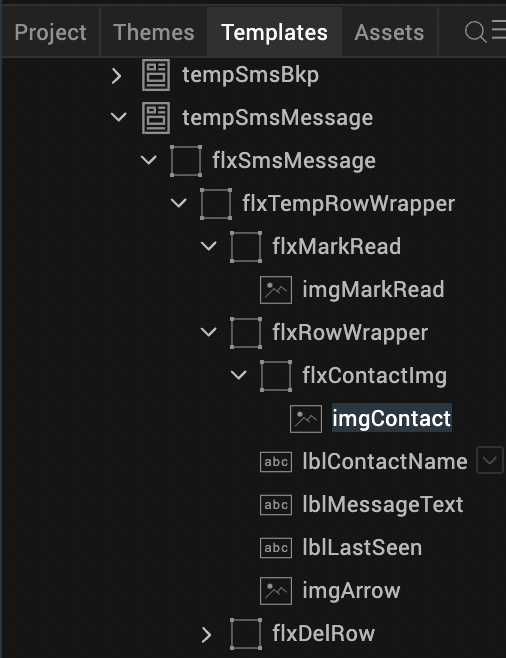
These are used to achieve the **Swipe to Delete/Mark** functionality.

* **flxMarkRead** is on the left side and **flxDelRow** is on the right side. They both contain one image widget each.
  + - * imgMarkRead - To hold the **Mark as Read** icon
      * imgDelRow - To hold the **Delete** icon

These are outside the device screen bounds.

* **flxRowWrapper** is in the middle and it holds the widgets visible inside the screen bounds.
  + - * It has a flex container on the left side, **flxContactImage**. This flex container holds an image widget, **imgContact** that holds the contact image.
      * It has three labels on the right side.
        + **lblContactName** - To show sender's name
        + **lblMessageText** - To show the received message
        + **lblLastSeen** - To show last seen
      * It also has an image on the right side, **imgArrow** to show an arrow image.

To change the **Mark Read** or **Delete** icons:

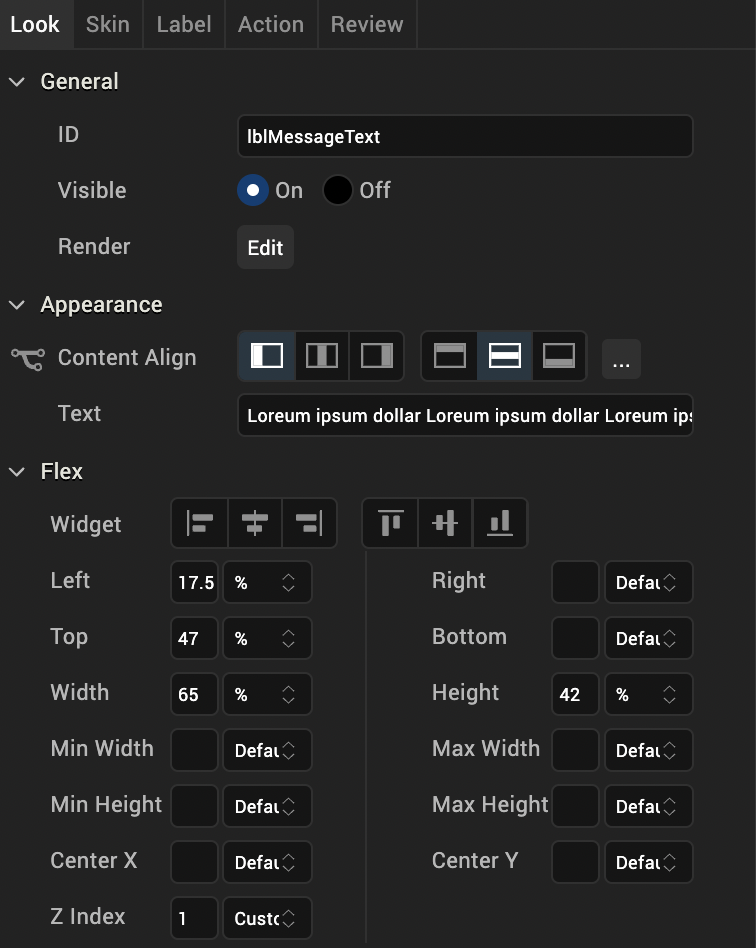
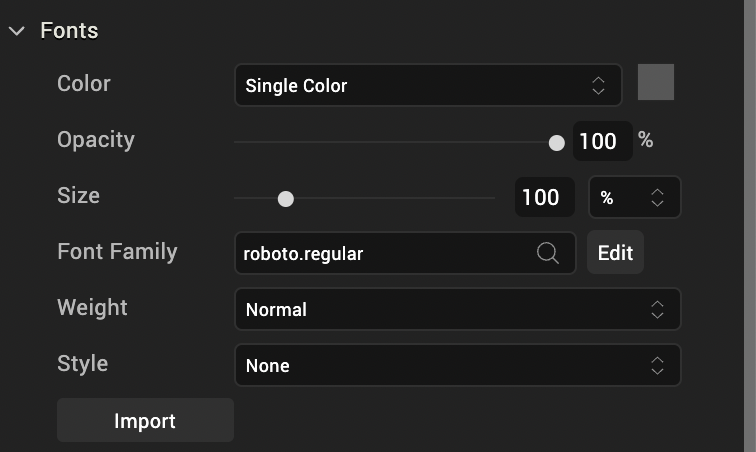
* + Under **Template**, select the image widget corresponding to the icon you want to change.  
    
  + Click on the **Image** tab on the **Properties** window on the right side.
  + Click on the **Edit** button next to the **Source** option.  
    A screenshot of a computer

    Description automatically generated with medium confidence
  + In the pop up window, select the image that you want to set and click **OK**.  
    A screenshot of a computer

    Description automatically generated

You can follow the same procedure to change any image

To modify the appearance of the **Labels**:

* + Under **Template**, select the label widget corresponding to the label you want to change.
  + Click on the **Look** tab in the **Properties** window on the right side.
  + In the **Look** tab, you can change the position of the label. This can be done by setting the properties for **left**, **right**, **top**, **bottom**, **centerX**, **centerY** and so on.  
    
  + Click on the **Skin** tab in the **Properties** window on the right side.
  + In the **Skin** tab, you can change the **font color**, **font family**, **size** and so on.  
    

Note: Make sure you import all the images that are needed under the **Assets** tab in the **Media** folder.

### **Product List**

The **Pull to Refresh** and **Expand-Collapse** screens use a segment to show a UI that you can create or modify under the **Templates** tab.

You can follow these guidelines to modify the **Template** for the **Product List UI**:

1. Import the app into Visualizer.
2. Go to the **Templates** tab in the **Project Explorer**.
3. In the provided list, expand **Mobile** > **Segments** > **tempProductList**
4. **tempProductList** is the template used in the **Product List UI**.
5. Expand **tempProductList** to explore the template.

A screenshot of a computer

Description automatically generated with medium confidence

As shown in the image, an outer flex container, **flxProductRow**, contains all the widgets used in the product list.

**flxProductRow** follows a **Flow Vertical** layout to implement the expand-collapse functionality. You can apply this layout under the A screenshot of a computer

Description automatically generated with medium confidence

**FlexContainer** tab in the **Properties** window on the right side.

**flxProductRow** contains a flex container **flxProductWrapper** that follows a **Flow Horizontal** layout. It also contains a label, **lblDetails** to hold the product description.

**flxProductWrapper** contains many widgets that make up the structure of the product details.

* + - We have a flex container on the left side, **flxImg** that contains an image widget, **img** to hold the product image.
    - We have another flex container on the right, **flxPro** that follows a **Flow Vertical** layout and it contains widgets to hold the product details.
    - **flxPro** contains four label widgets:
      * lblProductName - To show the product's name
      * lblDescription - To show a brief description about the product
      * lblCost - To show the price of the product
      * lblStock - To show stock availability of the product
    - **flxPro** also has a flex container below the labels, **flxImgMain**. It follows a **Flow Horizontal** layout with five image widgets to show the product rating.
    - **flxPro** has another flex container below the rating, **flxProductColors**. It follows a **Flow Horizontal** layout with three images to show the available product colors.

**Product List Mini**

The **Coupled Segments** screen uses to segments to compare product lists. These two segments have a UI that you can create or modify under the **Templates** tab.

You can follow these guidelines to modify the **Template** for the **Product List Mini UI**:

1. Import the app into Volt Iris.
2. Go to the **Templates** tab in the **Project Explorer**.
3. In the provided list, expand **Mobile** > **Segments** > **tempProductMini**
4. **tempProductMini** is the template used in the **Product List Mini UI**.
5. This template is a modified version of **tempProductList**.
6. The difference between these two templates is that **tempProductMini** doesn't have the label **lblDetails** in the outer flex container **flxProductRowMini**. It also does not have the label **lblDescription** in **flxProductWrapper**.

## A screenshot of a computer Description automatically generated with medium confidence

## Configuring the functionality of the Segment Feature App

### **Coupled Segments**

**To achieve the Coupled Segments functionality, do the following:**

1. Place two different segments, **segData1** and **segData2** inside a scroll flex container, **flxSegments**.
2. Use **tempProductMini** as the row template for both the segments.
3. Set the functionality that you want to provide:
   * To let the segments scroll individually, set the height of each segment as **100%** in the scroll flex container. Thus, any list of products that exceed this height will scroll within the flex.
   * To make the segments scroll together, keep the height of the segments as preferred in the scroll flex container. If the list of products is more than the visible area, the scroll container will scroll. As the top position of both segments is fixed inside the scroll flex container, they will scroll together.

## You can achieve the same functionality through code. You can add a switch, switchLock for switching between Lock and Unlock of the segments. The code can be written under frmCoupledSegments form controller.

Example:

|  |
| --- |
| onSwitchSlide:function(eventobject)  {  //check whether user enabled the Lock Segments switch  if(this.view.switchLock.selectedIndex == 1)  {  //if selected index is 1 it means user selected to unlock the segments  //write the logic to scroll segments individually here  }  else if(this.view.switchLock.selectedIndex === 0)  {  //if selected index is 0 it means user selected to lock the segments  //write the logic to scroll both segments here  }  this.view.forceLayout();  } |

### **Pull to Refresh**

**To achieve the Pull to Refresh functionality, do the following:**

1. Add a **segment** as direct child to a **form**.
2. Define the **onPull** action under the **Action** tab in the **Properties** window on the right side.  
   A picture containing text, screenshot, font, number

   Description automatically generated
3. Write the code in a callback function in the **frmPullToRefresh** form controller.

In the example code, the function shows an alert when the **onPull** action occurs.

|  |
| --- |
| segmentOnPullCallback:function()  {  alert("Segment Pull Triggered!!");  } |

### **Expand-Collapse**

**To achieve the Expand-Collapse functionality, do the following:**

1. Use the **tempProductList** template in a segment.
2. Initially assign data to all widgets but keep **lblDetails** empty.
3. Whenever the user clicks on a row, check the length of **lblDetails** of the **rowData**.

* If the length is zero, add product details to **lblDetails** of the **rowData** and re-assign row data to the segment.
* If the length is not zero, assign empty string to **lblDetails** of the **rowData** and re-assign row data to the segment.

You can achieve the same functionality through code. The code can be written under **productList** component controller.

|  |
| --- |
| // assign this function to onRowClick action of segment  productOnClick:function()  {  //retrieve row data of user selected row  var rowData = this.view.segProductList.selectedRowItems;  //check length of lblDetails in rowData  if(rowData[0].lblDetails.length ===0)  {  rowData[0].lblDetails = //assign product details here  }  else  {  //assign empty string to lblDetails here  rowData[0].lblDetails ="";  }  //re assign data to segment as shown below  var row1=  {  sectionIndex: this.view.segProductList.selectedRowIndex[0],  rowIndex : this.view.segProductList.selectedRowIndex[1]  };  this.view.segProductList.setDataAt(rowData[0],  this.view.segProductList.selectedRowIndex[1],  this.view.segProductList.selectedRowIndex[0]);  } |

### **Row Animation**

To achieve the Row Animation functionality, do the following:

1. Use **tempSmsMessage** as the row template for a segment.
2. Create an animation object to use with **addAll** or **setAnimations** APIs of the segment widget  
   For more information on animation objects, you can refer to the [Animation API Guide](http://docs.kony.com/konylibrary/visualizer/viz_api_dev_guide/Default.htm#animationapi.htm).

You can achieve the same functionality through code. The code can be written under the **smsMessage** component controller.

|  |
| --- |
| // call reFreshWithRowAnim function to set animations to segment  reFreshWithRowAnim:function()  {  var data = this.view.segMessagesList.data;  var addRowAnim=this.getAddRowAnim();  this.view.segMessagesList.addAll(data,addRowAnim);  this.onRowDisplayFunction();  },  getAddRowAnim:function()  {  var transformProp1 = kony.ui.makeAffineTransform();  transformProp1.scale(0,0);  var transformProp3 = kony.ui.makeAffineTransform();  transformProp3.scale(1,1);  var animDefinitionOne =  {  0:  {  "anchorPoint":{"x":0.5,"y":0.5},  "transform":transformProp1  },  100:  {  "anchorPoint":{"x":0.5,"y":0.5},  "transform":transformProp3  }  };  var animObj = kony.ui.createAnimation(animDefinitionOne);  var animConf =  {  delay:0,  fillMode:kony.anim.FILL\_MODE\_FORWARDS,  duration:0.7  };  var addRowAnimtion =  {  definition: animObj,  config: animConf,  callbacks: null  };  return addRowAnimtion;  },  onRowDisplayFunction:function ()  {  var animConfig =  {  "duration":0.3,  "iterationCount":1,  "delay":0,  "fillMode":kony.anim.FORWARDS  };  //scale  var transformProp1 = kony.ui.makeAffineTransform();  transformProp1.scale(0,0);  var transformProp3 = kony.ui.makeAffineTransform();  transformProp3.scale(1,1);  var animDefinitionOne =  {  0:  {  "anchorPoint":{"x":0.5,"y":0.5},  "transform":transformProp1  },  100:  {  "anchorPoint":{"x":0.5,"y":0.5},  "transform":transformProp3  }  };  var animDefinition = kony.ui.createAnimation(animDefinitionOne);  var finalAnimation =  {  definition: animDefinition,  config: animConfig  };  this.view.segMessagesList.setAnimations({visible: finalAnimation});  } |

### **Swipe to Delete/Mark**

Swipe to delete is achieved through the **smsMessage** component's exposed child widgets. These child widgets need to be exposed because the app is built on MVC Architecture.

In MVC Architecture, the **Form**, the **Component**, and the **Template** have their own **View**, **Controllers**, and **Life Cycle** as shown below.

If a **Form** uses a **Component**, it can access the child widgets and their properties only if it is exposed by the component.

To know more about MVC Architecture in Quantum, you can refer to the [Reference Architecture Guide](http://docs.kony.com/konylibrary/visualizer/kony_ref_arch_api/Default.htm).

**To achieve the Swipe to Delete/Mark functionality, do the following:**

1. Use **tempSmsMessage** as the row template for a segment. In this case, it is used in the **smsMessage** component.
2. Register a gesture recognizer on the row flex container, **flxSmsMessage** in **tempSmsMessage**, in the **template controller**.  
   This template should be registered everywhere this template is used. This is so that the **addGestureRecognizer** API is called in the template's life cycle event called **onViewCreated**.
3. The template is a child of the segment. This means that the **template controller** cannot handle the row of the segment.  
   Whenever a **swipe** is triggered and caught in the **template controller**, it should call the **executeOnParent** API in the **template controller**. This will notify the parent about the **swipe gesture** and the **row information**.
4. The code in the **template controller** is similar to the code given below.

|  |
| --- |
| /\*this is a template life cycle event  it will be called by the framework  whenever view for this template is created.  \*/  onViewCreated:function(eObj)  {  try  {  this.view.addGestureRecognizer(constants.GESTURE\_TYPE\_SWIPE,  {fingers: 1},  this.swipeGestureHandler);  }  catch(exc)  {  alert("exception in template!!!"+JSON.stringify(exc));  }  },  //swipe gesture handler method  swipeGestureHandler:function(widgetInfo,gestureInfo,context)  {  var mswipeDirection = gestureInfo.swipeDirection;  var secIndex = context["sectionIndex"];  var rowIndex = context["rowIndex"];  var myInfo =  {  section: secIndex,  row: rowIndex,  swipeDirection: mswipeDirection  };  this.executeOnParent("swipeDetected",myInfo);  } |

1. In the parent controller, define a method, **swipeDetected**. This will be called from the **template controller**'s swipe handler.
2. In the **swipeDetected** method, check whether the swipe direction is to the **left** or to the **right**. Check whether the current view is **Mark Read is revealed**, **default view**, or **Delete is revealed**.
3. Define the animation object according to the above values and perform the animation using the **animateRows** API of the segment.

|  |
| --- |
| swipeDetected: function(widgetInfo,swipeInfo)  {  try  {  var needToAnimate = false;  var curRowData = this.view.segMessagesList.data[swipeInfo.row];    //if current view = 1 -> default view  //if current view = 2 -> delete button is revealed  //if current view = 3 -> mark as read button is read  var vw = curRowData.curnt\_view;    // if swipe direction is 1 -> left swipe recognized  //if swipe direction is 2 -> right swipe recognized  var sd = swipeInfo.swipeDirection;  //define animation object  var fstStepConfig=  {  "left": "-20%",  "stepConfig":  {  "timingFunction": kony.anim.EASE  }  };  var lstStepConfig =  {  "left": "-20%",  "stepConfig":  {  "timingFunction": kony.anim.EASE  }  };    //inside all conditions modify the current row data to retain its view  if(sd==1 && vw==1)  {  // reveal del btn  fstStepConfig.left = "-20%";  lstStepConfig.left = "-40%";  curRowData.curnt\_view = 2;  curRowData.flxTempRowWrapper =  {  left : "-40%"  };  needToAnimate = true;  }    else if(sd==1 && vw==2)  {  // donothing  needToAnimate = false;  return;  }  else if(sd==1 && vw==3)  {  //change to default view from mark as read view  fstStepConfig.left = "0%";  lstStepConfig.left = "-20%";  curRowData.curnt\_view = 1;  curRowData.flxTempRowWrapper =  {  left : "-20%"  };  needToAnimate = true;  }    else if(sd==2 && vw==1)  {  //reveal mark read  fstStepConfig.left = "-20%";  lstStepConfig.left = "0%";  curRowData.curnt\_view = 3;  curRowData.flxTempRowWrapper =  {  left : "0%"  };  needToAnimate = true;  }    else if(sd==2 && vw==2)  {  //change to defautl view from delete view  fstStepConfig.left = "-40%";  lstStepConfig.left = "-20%";  curRowData.flxTempRowWrapper =  {  left : "-20%"  };  curRowData.curnt\_view = 1;  needToAnimate = true;  }    else if(sd==2 && vw==3)  {  //donothing  needToAnimate = false;  return;  }    if(needToAnimate)  {  this.view.segMessagesList.animateRows  ({  rows:  [{  sectionIndex : swipeInfo.section,  rowIndex : swipeInfo.row  }],    widgets:["flxTempRowWrapper"], // widget to be animated in template    animation:  {    definition:kony.ui.createAnimation(  {  "0" : fstStepConfig,  "100": lstStepConfig  }),  config:  {  "delay": 0,  "iterationCount": 1,  "fillMode": kony.anim.FILL\_MODE\_FORWARDS,  "duration": 0.5,  "direction": kony.anim.DIRECTION\_ALTERNATE  },    callbacks:  {  animationEnd : function()  {  //setting segment data so if any other row is in reveal state previously it will be reset to default  this.view.segMessagesList.setData(this.\_\_Initial\_Data);  //setting current row data to retain its position  this.view.segMessagesList.setDataAt(curRowData,swipeInfo.row);  }.bind(this)  }  });  }  }  catch(exc)  {  alert("exception in swipeHandler!!!!");  }  } |

1. Handling the **onClick** action of the delete button is done the same way as handling the **swipe** gesture. It should trigger a delete event from the **template controller** to the **parent controller**.
2. In the template controller, define a method, **onDeleteRow**. Call this method from the **onClick** action under the **Actions** tab in the **Properties** window on the right side.

the **onDeleteRow** method in the template controller should call the **executeOnParent** API to notify the parent controller about the **Delete** action.

|  |
| --- |
| onDeleteRow:function(eventobject,context)  {  var secIndex = context["sectionIndex"];  var rowIndex = context["rowIndex"];  this.executeOnParent("deleteRowDetected",  {  section:secIndex,  row:rowIndex  });  } |

In the parent controller, define a method **deleteRowDetected**. This will delete the row data from the segment using the **removeAt** API of the segment widget.

|  |
| --- |
| deleteRowDetected:function(widgetInfo,rowData)  {  this.view.segMessagesList.removeAt(rowData.row,rowData.section);  } |

1. **REVISION HISTORY**

App version 1.0.1

* 1. **Limitations**

No Limitations.

* 1. **Known issues**

The **Pull to Refresh** functionality is inconsistent on **iOS**.