Date : 23-Feb-22

Simple Sort

version: 1.0.4

# **Overview**

The Simple Sort component shows a set of options a user can choose to sort the content in your application. These include **Relevance**, **Low to High**, **High to Low**, and so on.

You can use the default sort options such as **Relevance** and **Low to High**, or you can add your own options to the component and retrieve them using the provided API. You can use the component in scenarios such as a retail app, where you want to provide the option to sort a list.

## **Use case:**

### The simple sort component provides a ready-made UI template for implementing a sort functionality applications. The components lists different sort options which can be modified as required. This type of component is generally used in retail, e-commerce, and other similar scenarios

### **B. Features**

### The simple search component provides a ready-made you a template for implementing a song function in applications.

### JSON input format

## **C. Percentage of re-use:**

70-80% (Data and skins are customized and can be changed manually)

# **Getting Started**

## **Prerequisites**

Before you start using the Simple Sort component, ensure you have the following:

1. [HCL Foundry](https://manage.hclvoltmx.com/)
2. Volt MX Iris

## **Platforms Supported**

### Mobile

#### iOS

#### Android

### Tablets

### PWA

## **Importing the Component**

## You can import the Forge components only into the apps that are of the Reference Architecture type.

## **To import the Simple Sort component, do the following:**

## Open your app project in Volt MX Iris.

1. In the Project Explorer, click the **Templates** tab.

Graphical user interface, text, application

Description automatically generated

3.Right-click **Components**, and then select **Import Component**. The **Import Component** dialog box appears.

Graphical user interface, text, application, Teams

Description automatically generated

4.Click **Browse** to navigate to the location of the component, select the component, and then click **Import**. The component and its associated widgets and modules are added to your project.

Text

Description automatically generated­

Once you have imported a component to your project, you can easily add the component to a form. For more information, refer [Add a Component to a Form](https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Iris/iris_user_guide/Content/C_UsingComponents.html#add-a-component-to-a-form).

## **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 Iris User Guide.

You can then run your app to see the Simple Sort work in real time.

# **References­**

## **Dynamic Usage**

If you want to use the Simple Sort component dynamically, you will need to import the component into your project Templates. Follow the given steps to do so

### Download the component from HCL Volt MX Marketplace as a zip file.

### Go to the Templates tab in your project explorer.

### Right click on Components and select Import Component.

### Navigate to where you downloaded your zip file and import it into Iris.

After you import the component into your project templates, you can add it to your app dynamically. To do so, follow the given steps

### Access the FormController of the form you want to add the component into.

### Create a function called createComponent(); and write the code inside it to create and configure the component.

### You can refer to the given sample code for more information.

/\*Creating a component's Object \*/

var SimpleSort=new com.voltmxmp.simplesort({

"clipBounds": true,

"height": "100%",

"id": "simplesort",

"isVisible": true,

"left": "0dp",

"top": "0dp",

"width": "100%",

"zIndex": 1

},{

},{});

//Configuring the component properties

SimpleSort.flxRoundedButtonSkn = "sknFlxRoundedBtn";

SimpleSort.sortIconSrc = "newsort.png";

SimpleSort.sortButtonText="Sort";

SimpleSort.sliderImageSrc="newdragger.png";

SimpleSort.optionSelectedSrc="acme.png";

SimpleSort.text="Sort by";

SimpleSort.optionSliderTop=30;

//creating API's

this.view.add(SimpleSort);

this.view.simplesort.getSortOptionSelected();

this.view.simplesort.sortOptions =

{

"data":

[

{"sortBy": "Relevance"},

{"sortBy": "Low-to-High"},

{"sortBy": "High-to-Low"},

{"sortBy": "What’s New"}

]

};

1. **Save** the file.

## **Properties**

The properties provided on the **Component** tab allow you to customize the UI elements in the **Simple Sort** component. You can set the properties directly on the **Component** tab or by writing a JavaScript.

**1. General**

**[Open](javascript:void(0);)1. SortByOptions (SortBy Options)**

|  |  |
| --- | --- |
| **Category:** | Pass Through |
| **Description:** | Specifies the sort options that you want to display on the component. | |
| **Syntax:** | sortOptions | |
| **Type:** | * Data Grid * JSON | |
| **Read/Write:** | Read + Write | |
| **Remarks:** | By default, the component contains two options:   * Relevance * Low to High | |
| **Example:** | this.view.componentID.sortOptions =  {  "data":  [  {"sortBy": "Relevance"},  {"sortBy": "Low-to-High"}  ]  }; | |

1. **Sort Icon Image (sortIconSrc)**

|  |  |
| --- | --- |
| **Category:** | Pass Through |
| **Description:** | Specifies the source of the image that you want to set as the [icon](javascript:void(0);) for the [Sort Button](javascript:void(0);). |
| **Syntax:** | sortIconSrc |
| **Type:** | String |
| **Read/Write:** | Read + Write |
| **Default Value:** | this.view.componentID.sortIconSrc = "sort\_icon.png"; |

1. **Sort Button Text(sortButtonText)**

|  |  |
| --- | --- |
| **Category:** | Pass Through |
| **Description:** | Specifies the [text](javascript:void(0);) that you want to display on the [Sort Button](javascript:void(0);). | |
| **Syntax:** | sortButtonText | |
| **Type:** | String | |
| **Read/Write:** | Read + Write | |
| **Default Value:** | "Sort" | |
| **Example:** | this.view.componentID.sortButtonText = "Sort"; | |

1. **Slider Image(sliderImageSrc)**

|  |  |
| --- | --- |
| **Category:** | Pass Through |
| **Description:** | Specifies the source of the image that you want to use on the [Slider](javascript:void(0);). |
| **Syntax:** | sliderImageSrc |
| **Type:** | String |
| **Read/Write:** | Read + Write |
| **Default Value:** | this.view.componentID.sliderImgSrc = "slider\_icon.png"; |

1. **Options Selected Image(optionsSelectedSrc)**

|  |  |  |
| --- | --- | --- |
| **Category:** | | Pass Through |
| **Description:** | Specifies the source of the image that you want to set in front of the [selected option](javascript:void(0);). | | |
| **Syntax:** | optionSelectedSrc | | |
| **Type:** | String | | |
| **Read/Write:** | Read + Write | | |
| **Default Value:** | "acme.png" | | |
| **Remarks:** | Make sure that the image that you want to set in front of the selected option exists in the *workspace*\*resources*\*common* directory. | | |
| **Example:** | this.view.componentID.optionSelectedSrc = "selected\_option.png"; | | |

1. **Sort options Top position in percentage(optionSliderTop)**

|  |  |  |
| --- | --- | --- |
| **Category:** | | Pass Through |
| **Syntax:** | optionSliderTop | | |
| **Type:** | Integer | | |
| **Read/Write:** | Read + Write | | |
| **Default Value:** | 30 | | |
| **Remarks:** | The component calculates this position from the top of the screen. Therefore, if this value is 30, then the top edge of the component is at 30% distance from the top of the screen. This means that covers 70% of the screen. | | |
| **Example:** | this.view.componentID.optionSliderTop = 30; | | |

**Sort Dimensions**

The following properties change the dimensions of the sort button.

1. **Center X**

|  |  |
| --- | --- |
| **Description:** | Specifies the [Center X](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm#Button_Properties.htm%23centerX) value that you want to set for the button. Center X determines the position of the horizontal center of the button. |
| **Syntax:** | buttonCenterX |
| **Type:** | String |
| **Read/Write:** | Read + Write |
| **Default Value:** | "50%" |
| **Remarks:** | You can specify the Center X value in **Dp**, **Px**, or **%**. |
| **Example:** | this.view.componentID.buttonCenterX = "50%"; |

1. **Width**

|  |  |
| --- | --- |
| **Description:** | Specifies the width that you want to set for the button. |
| **Syntax:** | buttonWidth |
| **Type:** | String |
| **Read/Write:** | Read + Write |
| **Default Value:** | "40%" |
| **Remarks:** | You can specify the width in **Dp**, **Px**, or **%**. |
| **Example:** | this.view.componentID.buttonWidth = "40%"; |

1. **Center Y**

|  |  |
| --- | --- |
| **Description:** | Specifies the [Center Y](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm#Button_Properties.htm%23centerY) value that you want to set for the button. Center Y determines the position of the vertical center of the button. |
| **Syntax:** | buttonCenterY |
| **Type:** | String |
| **Read/Write:** | Read + Write |
| **Default Value:** | "95%" |
| **Remarks:** | You can specify the Center Y value in **Dp**, **Px**, or **%**. |
| **Example:** | this.view.componentID.buttonCenterY = "95%"; |

1. **Height**

|  |  |
| --- | --- |
| **Description:** | Specifies the height that you want to set for the button. |
| **Syntax:** | buttonHeight |
| **Type:** | String |
| **Read/Write:** | Read + Write |
| **Default Value:** | "40dp" |
| **Remarks:** | You can specify the height in **Dp**, **Px**, or **%**. |
| **Example:** | this.view.componentID.buttonHeight = "40dp"; |

**2. Skins**

Skins define the appearance of a widget. You can apply and manipulate skins based on your preference. For more information, refer to the [Understanding Skins and Themes](http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Default.htm#Customizing_the_Look_and_Feel_with_Skins.htm) section of the Volt mx iris User Guide.

**Note:**  
Make sure that the skins that you assign to a **widget** are of the same **widget type**.

1. **Sort Button Skin**

|  |  |
| --- | --- |
| **Description:** | This skin links to the [Sort Button](javascript:void(0);). |
| **Syntax:** | flxRoundedButtonSkin |
| **Widget Type:** | [FlexContainer](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm#FlexContainer.htm) |
| **Example:** | this.view.componentID.flxRoundedButtonSkin = flexSortSkin; |

1. **Sort Text Skin**

|  |  |
| --- | --- |
| **Description:** | This skin links to the [Sort Label](javascript:void(0);). |
| **Syntax:** | sortTextSkin |
| **Widget Type:** | [Label](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm#label.htm) |
| **Example:** | this.view.componentID.sortTextSkin = labelSkin; |

1. **Sort Option Flex Skin**

|  |  |
| --- | --- |
| **Description:** | This skin links to the background of the [sort options](javascript:void(0);) popup. |
| **Syntax:** | flxSortOptionSkin |
| **Widget Type:** | [FlexContainer](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm#FlexContainer.htm) |
| **Example:** | this.view.componentID.flxSortOptionSkin = backgroundSkin; |

1. **Parent Flex Skin**

|  |  |
| --- | --- |
| **Description:** | This skin links to the background of the [parent container](javascript:void(0);). |
| **Syntax:** | sortParentFlxSkin |
| **Widget Type:** | [FlexContainer](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm#FlexContainer.htm) |
| **Example:** | this.view.componentID.sortParentFlxSkin = flexSkin; |

## **Events**

The component invokes events when its corresponding action is performed. You can configure any logic you want the component to perform whenever an event occurs. You can configure the events directly on the Actions tab or by writing a JavaScript, For more information, refer to [Add Actions](https://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Content/working_with_Action_Editor.htm) in the iris User Guide.

[**[Open](javascript:void(0);)1. onClickSortOption**](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Invoked when the user selects a sort option. |
| **Syntax:** | onClickSortOption |
| **Parameters:** | *selectedOption [String]* : The option that the user selected. |
| **Example:** | this.view.componentID.onClickSortOption = function(selectedOption)  {  alert("Selected Sort Option: "+selectedOption);  }.bind(this); |

## **API’s**

The following APIs pertain to the Simple Sort component:

[**[Open](javascript:void(0);)1. getSortOptionSelected**](javascript:void(0);)

|  |  |
| --- | --- |
| **Description:** | Retrieves the option that the user has selected on the component. |
| **Syntax:** | getSortOptionSelected() |
| **Parameters:** | None |
| **Return Value:** | *selectedOption [String]* : The option that the user selected. |
| **Example:** | var selectedOption = this.view.componentID.getSortOptionSelected();  alert("Selected Sort Option: "+selectedOption); |

# **Revision History**

App version 1.0.4

## **Known Issues**

No issues

## **B. Limitations**

No limitations