10-Jan-2022

sort and filter

version: 1.0.0

**1. OVERVIEW**

The Sort and Filter component helps us filter data present in a list based on criteria configured during the initialization of the component. Because all filters are dynamically generated, any type of filter is supported if a comparator/matcher can be provided for the same.

**A. Use case:**

This component can help design a filter functionality for data in a list/segment/sequence of flexes etc. It works directly with the data and not the structure in which the original data is displayed. As such, wherever a filter functionality is needed, this component can be used.

**B. Features:**

* Dynamically generated UI based on filters needed and structure of original data.
* Ability to work with any kind of listing structure in parent form.
* Any sort of complex filters/sorting criteria can be defined on the fly and there are no limits to how many can be configured or applied.

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

## 90%

# **2. Getting Started**

## 

## **A. Prerequisites**

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

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

## **B. Platforms Supported**

### Mobile

#### iOS

#### Android

### Tablets

### PWA

## **C. Importing the Component**

Before you start importing the component to Volt MX Iris, you must download the component from the HCL Volt MX website.

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

**To import the Sort and filter component, do the following:**

* 1. Open your app project in Volt MX Iris.
  2. In the project Explorer, click the Templates tab.

Graphical user interface

Description automatically generated with medium confidence

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

Graphical user interface, application, Teams

Description automatically generated

4. Click Browser 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.

Graphical user interface, text, application

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)

## **D. 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 Sort and filter work in real time.

# **3. References**

## **A. Dynamic Usage**

If you want to use the Sort and Filter 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 VoltMX 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. (filter functions are samples)

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

createComponent: function()

{

//Creating the component's object

var SortAndFilter = new com.voltmx.SortAndFilter(

{

"clipBounds": true,

"height": "100%",

"id": "SortAndFilter",

"isVisible": true,

"left": "100%",

"top": "0dp",

"zIndex": 90

},{},{});

//Configuring the component's properties

//Adding the component to the form

this.view.add(SortAndFilter);

this.view.SortAndFilter.setFilterOptions({

sortOptions : [

{

id:"sortOption1",

displayText : "Name (A-Z)",

comparator: function(a, b){

return (a.prodName.toUpperCase() < b.prodName.toUpperCase())?-1:a.prodName.toUpperCase() === b.prodName.toUpperCase()?0:1;

}

},

{

id:"sortOption2",

displayText : "Name (Z-A)",

comparator: function(a, b){

return (a.prodName.toUpperCase() < b.prodName.toUpperCase())?1:a.prodName.toUpperCase() === b.prodName.toUpperCase()?0:-1;

}

}

],

filterOptions: [

{

id: "filterOption1",

displayText:"Price",

filterSubOptions : [

{

id:"priceOption1",

displayText:"Under $100",

matcher: function(a) {

return a.prodPrice < 100 ;

}

},

{

id:"priceOption2",

displayText:"$100 - $500",

matcher: function(a) {

return a.prodPrice >= 100 && a.prodPrice <=500;

}

}

]

},

{

id: "filterOption2",

displayText:"Rating",

filterSubOptions : [

{

id:"rateOption1",

displayText:"1 & above",

matcher: function(a) {

return a.prodAvgUserReview >=1;

}

},

{

id:"rateOption2",

displayText:"2 & above",

matcher: function(a) {

return a.prodAvgUserReview >=2;

}

}

]

},

{

id: "filterOption3",

displayText:"On sale",

filterSubOptions : [

{

id:"onSaleOption1",

displayText:"true",

matcher: function(a) {

return a.onSale === "true";

}

},

{

id:"onSaleOption2",

displayText:"false",

matcher: function(a) {

return a.onSale === "false";

}

}

]

},

{

id: "filterOption4",

displayText:"Energy Star Rating",

filterSubOptions : [

{

id:"energyRatingOption1",

displayText:"1 star",

matcher: function(a) {

return a.quantityLimit === "1";

}

},

{

id:"energyRatingOption2",

displayText:"2 star",

matcher: function(a) {

return a.quantityLimit === "2";

}

}

]

},

{

id: "filterOption5",

displayText:"Color",

filterSubOptions : [

{

id:"colorOption1",

displayText:"Black",

matcher: function(a) {

return a.color ==="Black";

}

},

{

id:"colorOption2",

displayText:"Stainless steel",

matcher: function(a) {

return a.color ==="Stainless steel";

}

}

]

}

]

});

}

You can use a component's **Properties** to customize and configure the elements. These elements can be UI elements, service parameters, and so on. You can set the properties from the Iris's Properties panel on the right-hand side. You can also configure these properties using a JavaScript code.

## **B. Properties**

The properties provided on the **Component** tab allow you to customize the UI elements in the Sort and Filter component. You can set the properties directly on the **Component** tab or by writing a JavaScript. This section provides information about how to set the properties by writing a JavaScript.

## **C. 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 by writing a JavaScript.

**1. updateParentData**

|  |  |
| --- | --- |
| **Description:** | Invoked when the user taps or clicks a menu item. |
| **Syntax:** | updateParentData() |

**Example**: this.view.SortAndFilter.updateParentData=this.updateFiltereData;

updateFiltereData:function(){

this.view.segProduct.setData(this.view.SortAndFilter.getOperationalData()); }

## **D. APIs**

The following APIs pertain to the Sort and Filter component:

1. **setFilterOptions**

**Description**: Sets the FilterOptions for the form.

**Syntax**: setFilterOptions(data)

**Parameters**: sortOptions : [

{  
id:"sortOption1",  
displayText : "Name (A-Z)",  
comparator: function(a, b){  
return (a.prodName.toUpperCase() < b.prodName.toUpperCase())?-1:a.prodName.toUpperCase() === b.prodName.toUpperCase()?0:1;  
}  
}],filterOptions: [{  
id: "filterOption1",  
displayText:"Price",  
filterSubOptions : [{  
id:"priceOption1",  
displayText:"Under $100",  
matcher: function(a) {  
return a.prodPrice < 100 ;  
}}]},

{  
id: "filterOption2",  
displayText:"Rating",  
filterSubOptions : [{  
id:"rateOption1",  
displayText:"1 & above",  
matcher: function(a) {  
return a.prodAvgUserReview >=1;  
}}]}

**Example**: this.view.SortAndFilter.setFilterOptions({data});

1. **getOperationalData**

**Description**: Returns the OpertionalData.

**Syntax**: getOperationalData()

**Parameters**: None

**Example**: this.view.SortAndFilter.getOperationalData();

1. **setOperationlData**

**Description**: Used to set the Operational Data.

**Syntax**: setOperationlData (formReference)

**Parameters**: Data

**Example**: this.view.SortAndFilter.setOperationalData(sampleData)

1. **openFilter**

**Description**: Used to set the Data.

**Syntax**: openFilter (formReference)

**Parameters**: Data

**Example**: this.view.SortAndFilter. openFilter (sampleData)

# **4. Revision History**

App version 1.0.0:

## **A. Known Issues**

* If browser height is minimized, then UI can get distorted.