17 Sep 2024

**product list & details**

**version: 1.0.1**

# **Overview**

 A List Detail Interface is a common pattern in computer systems where a master list is displayed, and when an item is selected, more details about that item are displayed in a separate view. This List-Detail pattern caters to the e-commerce applications where all products are listed down in a single form and on click of any product the details page of the product is displayed.

## **Use case**

 Product List & Details is a highly configurable component that can be used in various commerce apps. Under the hood, the component uses [Object Services](https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Foundry/voltmx_foundry_user_guide/Content/Objectservices.html) on VoltMX Foundry to fetch data from back-end services.

You can use the component in scenarios such as: A retail app where you want to display a list of products and their information.

The component consists of two views:

* **List View**: Displays a list of all products. When a user selects a product, the component displays the Details View.
* **Details View**: Displays the details of the selected product.

.

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

80-85%

## **Features**

* + 1. List-Details layout pre-configured for displaying Products data
		2. Easily configurable- customize as per your requirement
		3. Connect with any backend data source

# **Getting Started**

## **Prerequisites**

Before you start using the Product List & Details component, ensure following:

• [HCL Foundry](https://manage.hclvoltmx.com/)

• Volt MX Iris

## **Platforms Supported**

### Mobile

#### iOS

#### Android

## **Importing the app**

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

##  **To import the Product List & Details component, do the following:**

## Open your app project in Volt MX Iris.

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

 

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



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.

##

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 User Guide.

 You can then run your app to see the  Product List & Details work in real time.

# **References**

## **Dynamic Usage**

You can also add Product List & Details component dynamically. To do so,

1. In the Project Explorer, on the Projects tab, click Controllers section to access the respective Form Controller. Create a method and implement the code snippet similar to the sample code mentioned below.

/\* Creating Product List & Details component instance \*/

/\* Creating the component's object \*/

var productList = new com.voltmxmp.productlist(

{

 "clipBounds": true,

 "height": "100%",

 "id": "productList",

 "isVisible": true,

 "layoutType": voltmx.flex.FREE\_FORM,

 "left": "0dp",

 "masterType": constants.MASTER\_TYPE\_USERWIDGET,

 "skin": "voltmxmpSknFlxBgBlue",

 "top": "0dp",

 "width": "100%",

}, {}, {});

/\* Setting component's properties \*/

productList.headerIsVisible = true;

productList.titleIsVisible = true;

productList.hamburgerIsVisible = true;

productList.titletext = "Clothes";

/\* Adding component to the form \*/

this.view.add(productList);

 In the code snippet, you can edit the properties of the component as per your requirement. For more information, see Setting Properties.

2. Save the file

## **Properties**

 The properties provided on the **Component** tab allows you to customize the elements in the  **Product List & Details** component. These elements can be UI elements, service parameters, and so on. You can set the properties from the Volt MX Iris Properties panel on the right-hand side. You can also configure these properties using a JavaScript code.

**General Properties**

**1.** **Header Visibility**

| **Description:** | Toggles the visibility of the header.. |
| --- | --- |
| Syntax: | headerIsVisible |
| **Type:** | Boolean |
| **Read/Write:** | Read + Write |
| **Example:** | this.view.componentID.headerIsVisible = true; |
| **Default Value:** | true |

**2. Hamburger Visibility**

| **Description:** | Toggles the visibility of the hamburger menu.. |
| --- | --- |
| Syntax: | hamburgerIsVisible |
| **Type:** | Boolean |
| **Read/Write:** | Read + Write |
| **Example:** | this.view.componentID.hamburgerIsVisible = true; |
| **Default Value:** | true |

 **3. Title Visibility**

| **Description:** | Toggles the visibility of the title. |
| --- | --- |
| Syntax: | titleIsVisible |
| **Type:** | Boolean |
| **Read/Write:** | Read + Write |
| **Example:** | this.view.componentID.titleIsVisible = false; |
| **Default Value:** | false |

**4. Search Visibility**

| **Description:** |  Toggles the search option in the component. |
| --- | --- |
| Syntax: | searchIsVisible |
| **Type:** | Boolean |
| **Read/Write:** | Read + Write |
| **Example:** | this.view.componentID.searchIsVisible = false; |
| **Default Value:** | false |

**5****.** **More Option Visibility**

| **Description:** |  Toggles the visibility of the menu icon. |
| --- | --- |
| Syntax: | moreIsVisible |
| **Type:** | Boolean |
| **Read/Write:** | Read + Write |
| **Example:** | this.view.componentID.moreIsVisible=true; |
| **Remarks:** | The default value of the property is **true**. |

**6. Title Text**

| **Description:** |  Specifies the text that you want to display as the title. |
| --- | --- |
| Syntax: | this.view.componentID.titleText="Mobiles"; |
| **Type:** | String |
| **Read/Write:** | Read + Write |
| **Example:** | this.view.componentID.titleText = "Products" |

**B. Skins Section**

1. **Header S****kin**

| **Description:** |  This skin links to the header  |
| --- | --- |
| **Syntax:** | sknHeader |
| **Widget Type:** | [FlexContainer](https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Iris/iris_widget_prog_guide/Content/FlexContainer.html) |
| **Example:** | this.view.componentID.sknHeader = "skin\_name"; |

## **API’s**

###   **1. fetchData**

| **Description:** | Fetches the data from the back-end service using the specified Object Service. |
| --- | --- |
| **Syntax:** | fetchData(objectService,dataModelObject,queryParams,successCallback,errorCallback) |
| **Parameters:** | * *objectService [String]*:The name of the Object Service that you want to use to fetch the data.
* *dataModelObject [String]*:The name of the data model that you want to use to fetch the data.
* *queryParams [JSON]*:Any parameters that you want to pass with the query.
* *successCallback [Function]*:An event that is invoked after the component fetches the data.
* *errorCallback [Function]*:An event that is invoked if any error occurs while fetching the data.
 |
| **Example:** | var objectService = "productDB";var dataModelObject = "category";var queryParams= {};var successCallback = function(response){ alert("Data fetched successfully");}.bind(this);var errorCallback = function(error){ alert("An error occurred while fetching the data.");}.bind(this);this.view.componentID.fetchData(objectService, dataModelObject,queryParams, successCallback, errorCallback); |

###   **2. insertRecords**

| **Description:** | Inserts the data into the back-end storage using the specified Object Service. |
| --- | --- |
| **Syntax:** | insertRecords(objectService,dataModelObject,queryParams,successCallback,errorCallback) |
| **Parameters:** | * *objectService [String]*:The name of the Object Service that you want to use to insert the data.
* *dataModelObject [String]*:The name of the data model that you want to use to insert the data.
* *queryParams [JSON]*:Any parameters that you want to pass with the query.
* *successCallback [Function]*:An event that is invoked after the component inserts the data.
* *errorCallback [Function]*:An event that is invoked if any error occurs while inserting the data.
 |
| **Example:** | var objectService = "productDB";var dataModelObject = "category";var queryParams= {"categoryDescription": "Puma product category", "categoryId": "512", "categoryName": "Puma", "categoryParentId": "487", "CreatedBy": "", "LastUpdatedBy": ""};var successCallback = function(response){ alert("Data inserted successfully");}.bind(this);var errorCallback = function(error){ alert("An error occurred while inserting the data.");}.bind(this);this.view.componentID.insertRecords(objectService, dataModelObject,queryParams, successCallback, errorCallback); |

### **3. fetchSuccessCallback**

| **Description:** | Fetches data from the back-end storage and inserts the data into the component. |
| --- | --- |
| **Syntax:** | fetchSuccessCallback(response) |
| **Parameters:** | *response [JSON]*:The response from VoltMX Foundry. |
| **Remarks:** | You can use this API to handle the success callback for the fetchData API. |
| **Example:** | var objectService = "productDB";var dataModelObject = "category";var dataFields = {};var errorCallback = function(error){ alert("An error occurred while fetching the data.");}.bind(this); //Using the API as a success callbackvar successCallback = this.view.componentID.fetchSuccessCallback.bind(this);this.view.componentID.fetchData(objectService, dataModelObject,dataFields , successCallback, errorCallback); |

### **4.fetchErrorCallback**

| **Description:** | Dismisses the loading indicator and displays an alert with information about the error. |
| --- | --- |
| **Syntax:** | fetchErrorCallback(error) |
| **Parameters:** | *error [JSON]*:Information about the error such as the error code and error message. |
| **Remarks:** | You can use this API to handle the error callback for the fetchData API. |
| **Example:** | var objectService = "productDB";var dataModelObject = "category";var queryParams= {};var successCallback = function(response){ alert("Data fetched successfully");}.bind(this);//Using the API as an error callbackvar errorCallback = this.view.componentID.fetchErrorCallback.bind(this);this.view.componentID.fetchData(objectService, dataModelObject,queryParams, successCallback, errorCallback); |

###

### **5. setDataToList**

| **Description:** | Adds the provided data to the List View of the component. |
| --- | --- |
| **Syntax:** | setDataToList(productListData, widgetDataMap) |
| **Parameters:** | * *productListData [Array of JSON]*:The data that you want to add to the list.
* *widgetDataMap [JSON]*:The data mapping of the back-end objects to the front-end widgets.
 |
| **Example:** | var productListData =[{ //Response from VoltMX Foundry}];var widgetDataMap ={ "imgThumbnailProduct": "img", "lblBrandNameList": "productName", "lblPriceList": "listCost", "lblRating": "rating", "lblBrandDetailsList": "shortDescription", "imgStar1": "img1", "imgStar2": "img2", "imgStar3": "img3", "imgStar4": "img4", "imgStar5": "img5",};this.view.componentID.setDataToList(productListData, widgetDataMap); |

###

### **D. Events**

### **1. onClickMoreMenu**

| **Description:** | Invoked when the user selects the menu option. |
| --- | --- |
| **Syntax:** | onClickMoreMenu() |
| **Example:** | this.view.componentID.onClickMoreMenu = function(){ alert("Menu Clicked");}.bind(this); |

### **2. onClickHamburger**

| **Description:** | Invoked when the user selects the hamburger menu icon. |
| --- | --- |
| **Syntax:** | onClickHamburger() |
| **Example:** | this.view.componentID.onClickHamburger = function(){ alert("Hamburger Clicked");}.bind(this); |

### **3. onSearchbarTextChange**

| **Description:** | Invoked when the user types any text in the search bar. |
| --- | --- |
| **Syntax:** | onSearchbarTextChange() |
| **Example:** | this.view.componentID.onSearchbarTextChange = function(){ alert("Search Bar Text Changed");}.bind(this); |

### **4. searchOnDone**

| **Description:** | Invoked when the user clicks on **Done** key in the keyboard |
| --- | --- |
| **Syntax:** | searchOnDone() |
| **Example:** | this.view.componentID.searchOnDone = function(){ alert("Searching...");}.bind(this); |

### **5. onClickAddToCart**

| **Description:** | Invoked when the user clicks or taps the Add To Cart button in the Details View. |
| --- | --- |
| **Syntax:** | onClickAddToCart() |
| **Example:** | this.view.componentID.onClickAddToCart = function(){ alert("Item added to cart.");}.bind(this); |

### **6. fetch**

| **Description:** | This event is used when a user wants to make network calls to fetch the data for the Product List-Details component. |
| --- | --- |
| **Syntax:** | fetch() |
| **Example:** | this.view.componentID.fetch = function(){ alert("fetching the data");}.bind(this); |

## **5. Configuring the Foundry App**

### When you import the Product List-Details component into VoltMX Iris, a VoltMX Foundry app (with name same as your VoltMX Iris Project Name) is also uploaded to your VoltMX Foundry account.

### **Note:**

### The name of the VoltMX Foundry app is the same as your VoltMX Iris Project Name.

### The VoltMX Foundry app fails to upload if already any app exists with the same name.

### Using the VoltMX Foundry console, you can configure the app-related orchestration and object services, and then publish the app.

### The following sections help you use the VoltMX Foundry.

### Importing Sample data

### Configuring VoltMX Foundry Services

### Mapping Product Data Model (Object services) with a back end Data Source

### Modifying the Data Model

### **Importing Sample data**

### Sample product data is bundled with the Product List-Details component. The sample data is available in the productDB.zip file located in your Visualizer workspace at \<works pace>\<project name>\resources\mobile\common\raw\ productDB.zip

### **To import the sample data, do the following:**

### Log on to your [VoltMX Foundry account](https://manage.hclvoltmx.com/). The **Dashboard** page appears by default.

### In the left pane, click the **Apps** menu. The **Custom Apps** page appears.

### Find and click the ListDetail (your project's name) app. .

### Click the **Publish** tab. The **Publish** tab opens. You must publish the app at least once before importing the sample data.

### In the **Publish** tab, In your environment box, in the **RUNTIME CONSOLES** section, click the object services icon (highlighted in the following figure). The Object Services page appears.

### Find the **productDB** object service by typing in the app name search box. Click import icon (highlighted in the following figure) and select the productDB.zip file located in your VoltMX Iris workspace at \<works pace>\<project name>\resources\mobile\common\raw\productDB.zip

###

###

###

###

### **Configuring VoltMx Foundry Services**

### The product VoltMX Foundry app contains a set of orchestration, and objects services by default. Using the VoltMX Foundry console, you can view and modify these services and also create new services.

### **Orchestration Service**

### Orchestration services are used to integrate multiple services and form a single service. The **productOrchestration** orchestration service fetches all the data objects in the mobile app in one go (to reduce the number of service calls). If you don't want to fetch all the data objects in one go, then delete the **productOrchestration** orchestration service (and the **productionListOS** which is a service driven object or wrapper on **productDB** orchestration service)

### **To view the orchestration services, do the following**:

### After logging on to your VoltMX Foundry account, open the **product**(your project's name) app.

### Click the **Orchestration** sub-tab. The **Orchestration** sub-tab opens with a list of services.

### Click on the **productOrchestration** service. The **Service Definition** tab of the selected orchestration service opens by default.

### Click the Operations tab to view the operations of the selected service.**productDB** service contains the **productOrchestration** operation. The operation fetches and combines all the data objects from **productDB**.

### **Object Services**

### The object services help you define the app data model and map it with relevant back end services. In the object service, you can view all the objects of the data model and the default fields and relationships of each object. To learn more about object services and mapping, see [Object Services](https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Foundry/voltmx_foundry_user_guide/Content/Objectservices.html).

### By default, the Product List-Details component is mapped with the VoltMX Foundry storage. The **productDB** is the data model schema for the VoltMX Foundry data storage.

### The **product** VoltMX Foundry app contains the following object services:

### **productDB** service contains the Product Data Model

### **productListOS** service acts as a service driven object for the orchestration service (wrapper around the orchestration service **productOrchestration**).

### **To view the object service and its operations, do the following**:

### After logging on to your VoltMX Foundry account, open the **product** VoltMX Foundry app.

### Click the **Objects** sub-tab. The **Objects** sub-tab opens with a list of data models.

### Click any one of the data models. For example, **productDB**. The **Service Definition** tab of the selected service opens by default.

### **Mapping Product List-Details Data Model (Object services) with a back end Data Source**

### Product List-Details component comes pre-configured with Product Data Model (by default mapped to VoltMX Foundry Storage). The data model contains multiple objects and fields related to product information. The following diagram illustrates the objects in the data model and relationships among them.

### The app data model is created as an object service. Using the VoltMX Foundry console, you can modify the fields and relationships of the objects in the data model. For example, you can add new fields to objects and define new relationships between objects.

### By default, the **product** VoltMX Foundry app is mapped to the VoltMX Foundry storage. You can easily decouple the app data model with the VoltMX Foundry storage and map it with other back end data sources.

### **To map with a back end data source, do the following:**

### On the **product** VoltMX Foundry app page, click the **Configure Services** tab, and then click **Objects** sub-tab. The Objects sub-tab opens along with the **productD** object service listed.

### Click the context menu icon of **productDB** as shown in the following figure. The context menu opens.

### Click **Clone App Data Model**. A copy of Product Data Model is created and the **New Object Service** page appears.

### In the **Name** box, type a name for a copy of the Product Data Model.

### Click the context menu arrow to expand the **Endpoint** Type list. A drop-down list opens.

### Select the required back end data source that you want to map and click **SAVE & CONFIGURE**.

### **Modifying the Data Model**

### The **productDB** object service allows you to modify the app data model as you require. You can edit(or)delete(or)add objects and fields in the data model based on your requirement. Every object in the object service contains the following sections:

### **Fields**

### In the Fields section of an object, you can add new fields and edit and delete the fields associated with the object. Every object has a set of fields that you cannot edit or delete. For example, the **CreatedBy** and **CreatedDataTime** fields of the **user** object cannot be modified.

### **Relationships**

### In the Relationships section of an object, you can define new relationships with other objects and edit and delete existing relationships of the object.

### For more information, refer to [Object Services](https://opensource.hcltechsw.com/volt-mx-docs/docs/documentation/Foundry/voltmx_foundry_user_guide/Content/Objectservices.html).

### After making the modifications in the app data model you also need to change the data object mappings with UI widgets. Refer to the steps mentioned in Mapping data from Object Services section.

### The app fetches all the data at once (to reduce the number of service calls) using the **productOrchestration** operation of the **productListOS**. The **productListOS** is a service driven object (wrapper) of the **productOrchestration** orchestration service.

### If you do not want the mobile client to fetch all the data at once, you can delete the services and directly bind the data objects of **productDB** and associate the data objects to the UI of respective widgets.

### However, If you want to fetch all the data at once on the client side, then after adding/deleting an object from the app data model you need to add/delete that object in the orchestration service **productOrchestration**.

### **Note:** You can also delete the existing Product Data Model and use a completely new app data model. In that case, you need to associate the new object services with the UI. Refer to the steps are mentioned in Mapping data from Object Services section.

##

##

##

##

##

##

##

##

##

##

## **6. Mapping UI with Data from Object Services**

### You can easily map UI widgets with object services. For example the following image illustrates the process of mapping the data to the List view segment.

###

### As shown above:

### The List view consists of thumbnail, BrandNameList, Rating, PriceList, and BrandDetailsList widgets.

### mediaContent, productName, ratingValue, listCost and shortDescription are the corresponding object services in the VoltMX Foundry. app, which is associated with Visualizer app.

### Here is a sample code which is used to map the widgets with respective object services.

### productList["thumbnail"] = media["mediaContent"];

### productList["lblBrandNameList"] = product["productName"];

### productList["lblRating"] = rating["ratingValue"];

### productList["lblPriceList"] = listPrice["listCost"];

### productList["lblBrandDetailsList"] = product["shortDescription"];

### **Note:** The app fetches all the data at once (to reduce the number of service calls) using the **productOrchestration** operation of the **productListOS**. The **productListOS** is a service driven object (wrapper) of the **productOrchestration** orchestration service. If you do not want the mobile client to fetch all the data at once, you can delete the services and directly bind the data objects of **productDB** and associate the data objects to the UI of respective widgets.

## **7. Modifying the Product List-Details UI**

### You can easily modify the UI of the Product List-Details component to suit your requirements.

### **Example**:

### The following procedures explain how to modify UI elements.

### **To modify the List view UI, do the following:**

### In the Project explorer, click the **Template** tab. A list of templates is displayed.

### Go to **Mobile** >> **Segments** >> datacontainer (**segRowTemplate**)

### Modify/ Delete/ Add widgets to the datacontainer template as per your requirement.

### You can also configure data mapped to a widget. See Mapping UI with Data from Object Services.

### The following figures illustrate how to remove the **lblBrandDetailsList** Label Widget from the row template.

###

### **To modify the Details view, do the following:**

### In the Project explorer, click the **Template** tab. A list of templates is displayed.

### Go to **Components** >> **com.voltmxmp.listdetail** >> **listdetail** >> **flxDetailsScreen**

### Modify/ Delete/ Add widgets as per your requirement.

### You can also configure the data mapped to a widget. See Mapping UI with Data from Object Services.

### The following figure illustrates how to re-arrange the **flxProductInfo** flex.

###

### The following figure reflects the **flxProductInfo** flex after the **Move Down** functionality.

###

### Similarly, you can delete other widgets or rows. If you wish to add new rows in details view, you can directly copy one of the existing flexes or add a new flex. The Product List-Details component also comes with a few additional flexes. To use the additional flexes, turn on the visibility of the required flexes.

# **Revision History**

 App version 1.0.1

## **Limitations**No Limitations