Date: 11 April 2023

Dynamic Form

VERSION: 1.0.1

# **Overview**

Dynamic Form is a web component that contains text fields to receive inputs from a user and contains submit button.

## **Use case:**

You can customize the data and the appearance of the fields that you want to display on the component. You can use the component in scenarios such as: a restaurant app, in which you want to provide an option to the user to contact the restaurant to provide feedback or suggestions.

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

80-90%

## **Features:**

* Initiate your app tasks
* Add actions
* Facility to customize the UI elements

# **Getting Started**

## **Prerequisites:**

Before you start using the Dynamic From component, ensure the following:

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

• Volt MX Iris

## **Platforms Supported:**

### PWA

## **Importing the app:**

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

## **To import the Dynamic Form 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

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

Graphical user interface, text, application, Teams

Description automatically generated

1. 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" \l "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 Dynamic From work in real time.

# **References**

## **Dynamic Usage:**

You can also add **Dynamic Form** 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.



createComponent: function()

{

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

var DynamicForm = new com.voltmxmp.dynamicform(

{

"clipBounds": true,

"height": "100%",

"id": "DynamicForm",

"isVisible": true,

"left": "0dp",

"top": "0dp",

"width": "100%",

"zIndex": 1

}, {}, {});

/\* Setting the component's properties \*/

DynamicForm.masterData =

{

"data":

[

{

"fieldId" : "name",

"fieldLabel" : "Name",

"fieldValue" : "",

"fieldType" : "text"

},

{

"fieldId" : "email",

"fieldLabel" : "Email",

"fieldValue" : "",

"fieldType" : "text"

},

{

"fieldId" : "phone",

"fieldLabel" : "Phone Number",

"fieldValue" : "",

"fieldType" : "text"

},

{

"fieldId" : "address",

"fieldLabel" : "Address",

"fieldValue" : "",

"fieldType" : "textarea"

}

]

};

DynamicForm.fieldSpace = 10;

DynamicForm.submitText = "Submit";

/\* Adding the component to the Form \*/

this.view.add(DynamicForm);

}

To execute the code, you need to call the createComponent function using **Actions**. For more information, refer to the [Add Actions](http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Default.htm#working_with_Action_Editor.htm?TocPath=Designing%2520an%2520Application|Add%2520Actions|_____0) section of the Iris User Guide.

2. Save the file

## **Properties:**

You can use a component's **Properties** to customize and configure the elements such as UI widgets, service parameters, and other elements. For more information about properties, you can refer to the [Components Overview](http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Default.htm#C_ComponentsOverview.htm?TocPath=Creating%2520Applications%2520with%2520Components|_____1) section of the Iris User Guide.

You can set the properties from the **Properties** panel on the right side of Iris. You can also configure these properties by using a JavaScript code.

**1. General**

**1. Form Fields**

|  |  |
| --- | --- |
| **Description:** | Specifies the data of the fields that you want to display on the component. |
| Syntax: | masterData |
| **Type:** | * Data Grid * JSON |
| **Read/Write:** | Write |
| **Example:** | this.view.componentID.masterData =  {  "data":  [  {  "fieldId" : "name",  "fieldLabel" : "Name",  "fieldValue" : "",  "fieldType" : "text"  },  {  "fieldId" : "email",  "fieldLabel" : "Email",  "fieldValue" : "",  "fieldType" : "text"  },  {  "fieldId" : "phone",  "fieldLabel" : "Phone Number",  "fieldValue" : "",  "fieldType" : "text"  },  {  "fieldId" : "address",  "fieldLabel" : "Address",  "fieldValue" : "",  "fieldType" : "textarea"  }  ]  }; |
| **Remarks:** | Every element in the data grid contains the following keys:   * fieldId *[String]*: A unique identifier for the field. * fieldLabel *[String]*: The text that the component displays as the title of the field. * fieldValue *[String]*: The text that the component displays inside the field. * fieldType *[String]*: The type of widget that you want to use for the field. Supports the following widgets:   + *text* : [TextBox widget](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm" \l "TextBox.htm" \t "_blank)   + *textarea* : [TextArea widget](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm" \l "TextArea.htm" \t "_blank) |

**2. Space between fields in dp**

|  |  |
| --- | --- |
| **Description:** | Specifies the space (in dp) between every field. |
| Syntax: | fieldSpace |
| **Type:** | Integer |
| **Read/Write:** | Write |
| **Example:** | this.view.componentID.fieldSpace = 10; |
| **Default value:** | 10 |

**3. Submit Text**

|  |  |
| --- | --- |
| **Description:** | Specifies the text that you want to display on the submit button. |
| Syntax: | submitText |
| **Type:** | String |
| **Read/Write:** | Read + Write |
| **Example:** | this.view.componentID.submitText = "Submit"; |
| **Default value:** | Submit |

## **Skins**

Skins define the appearance of a widget. You can apply and customize 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 VoltMX Iris User Guide.

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

**4. Field Flex Container**

|  |  |
| --- | --- |
| **Description:** | This skin links to the background of the fields. |
| Syntax: | sknDfFieldFlx |
| **Widget Type:** | [Flex Container](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm#FlexContainer.htm) |
| **Example:** | this.view.componentID.sknDfFieldFlx = "fieldBackgroundSkin"; |

**5. Input Label**

|  |  |
| --- | --- |
| **Description:** | This skin links to the field label. |
| Syntax: | sknDfLbl |
| **Widget Type:** | [Label](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm#Label.htm) |
| **Example:** | this.view.componentID.sknDfLbl = "fieldLabelSkin"; |

**6. Text Input**

|  |  |
| --- | --- |
| **Description:** | This skin links to the field text box. |
| Syntax: | sknDfTextInput |
| **Widget Type:** | [Text Box](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm#TextBox.htm) |
| **Example:** | this.view.componentID.sknDfTextInput = "fieldTextBoxSkin"; |

**7. Button Submit**

|  |  |
| --- | --- |
| **Description:** | This skin links to the submit button. |
| Syntax: | sknDfSubmitBtn |
| **Widget Type:** | [Button](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm#Button.htm) |
| **Read/Write:** | Read + Write |
| **Example:** | this.view.componentID.sknDfSubmitBtn = "submitButtonSkin"; |

**8. Submit Button Focus**

|  |  |
| --- | --- |
| **Description:** | This skin links to the submit button when it is in focus. |
| Syntax: | sknDfSubmitBtnFocus |
| **Widget Type:** | [Button](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm#Button.htm) |
| **Read/Write:** | Read + Write |
| **Example:** | this.view.componentID.sknDfSubmitBtnFocus = "submitButtonFocusSkin"; |

**9. Textarea input**

|  |  |
| --- | --- |
| **Description:** | This skin links to the field text area. |
| Syntax: | sknDfTextAreaInput |
| **Widget Type:** | [Text Area](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm#TextArea.htm) |
| **Read/Write:** | Read + Write |
| **Example:** | this.view.componentID.sknDfTextAreaInput = "textAreaSkin"; |

**10. Normal**

|  |  |
| --- | --- |
| **Description:** | This skin links to the background of the submit button. |
| Syntax: | sknDfSubmitButton |
| **Widget Type:** | [Flex Container](https://docs.kony.com/konylibrary/visualizer/viz_widget_prog_guide/Default.htm#FlexContainer.htm) |
| **Read/Write:** | Read + Write |
| **Example:** | this.view.componentID.sknDfSubmitButton = "submitButtonBackgroundSkin"; |

## **APIs**

## The following APIs pertain to the Dynamic Form component:

#### **setData**

|  |  |
| --- | --- |
| **Description:** | Creates fields on the component based on the specified data. |
| Syntax: | setData(formData) |
| **Parameters:** | *formData [Array of JSON]*: The data of the fields that you want to display on the component. |
| **Return Value:** | None |
| **Example:** | var formData =  [  {  "fieldId" : "name",  "fieldLabel" : "Name",  "fieldValue" : "",  "fieldType" : "text"  },  {  "fieldId" : "email",  "fieldLabel" : "Email",  "fieldValue" : "",  "fieldType" : "text"  },  {  "fieldId" : "phone",  "fieldLabel" : "Phone Number",  "fieldValue" : "",  "fieldType" : "text"  },  {  "fieldId" : "address",  "fieldLabel" : "Address",  "fieldValue" : "",  "fieldType" : "textarea"  }  ];  this.view.componentID.setData(formData); |

## **Events**

The component invokes events when its corresponding action is performed. You can configure logic that you want the component to perform when an event occurs.

You can configure the events on the **Actions** tab in the **Properties** panel. You can also configure the events by using a JavaScript code. 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.

### **1. onSubmit**

|  |  |
| --- | --- |
| **Description:** | Invoked when the user clicks the submit button. |
| Syntax: | onSubmit |
| Parameters: | *formValues [Array of JSON]*: The IDs and values of the fields of the component. |
| Sample JSON: | [  {  "fieldId":"dftextInputname",  "fieldValue":""  },  {  "fieldId":"dftextInputemail",  "fieldValue":""  },  {  "fieldId":"dftextInputphone",  "fieldValue":""  },  {  "fieldId":"dftextareaInputaddress",  "fieldValue":""  }  ] |
| **Example:** | this.view.componentID.onSubmit = function(formValues)  {  alert("Form Data: "+JSON.stringify(formValues));  }.bind(this); |

### **2. onErrorCallback**

|  |  |
| --- | --- |
| **Description:** | Invoked when an error occurs within the component. |
| Syntax: | onErrorCallback |
| Parameters: | *errObj [JSON]:* Information about the error, such as the error code and the error message. |
| **Example:** | this.view.componentID.onErrorCallback = function(errObj)  {  alert("Error Occured: "+JSON.stringify(errObj));  }.bind(this); |

# **Revision History**

App version 1.0.1:

## Limitations

1. Landscape mode is not supported.

2. If browser height is minimized UI gets distorted.